FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Analýza nálady v komunikačních aplikacích
Student:	Bc. Tomáš Valenta
Vedoucí:	Ing. Pavel Švagr
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je vytvořit nástroj na analýzu vývoje nálady na komunikačních platformách (Slack, Discord) pomocí sentimentové analýzy

1. Analyzujte aktuální používané metody a nástroje pro určení sentimentu [1][2].
2. Navrhněte metriky pro měření vývoje nálady v čase.
3. Vyberte vhodný model pro analýzu dat v čase a pokuste se predikovat budoucí vývoj nálady [3][4].
4. Navržené řešení naimplementujte, porovnejte jednotlivé metody sentimentové analýzy.
5. Vytvořte jednoduchou integraci implementovaného nástroje do zvolené platformy (min. pro Slack), která poskytne denní report do jednotlivých kanálů.
6. Otestujte na reálných datech z firemního prostředí poskytnuté vedoucím práce.

[1] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey, 2014

[2] Ronen Feldman. Techniques and applications for sentiment analysis. Commun. ACM, 56(4):82–89, apr 2013.

[3] R.J. Hyndman & G. Athanasopoulos: Forecasting: Principles and Practice.

[4] C. Chatfield: The Analysis of Time Series: An Introduction.

Elektronicky schválil/a Ing. Magda Friedjungová, Ph.D. dne 6. listopadu 2022 v Praze.

Diplomová práce

ANALÝZA NÁLADY V KOMUNIKAČNÍCH APLIKACÍCH

Bc. Tomáš Valenta

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: Ing. Pavel Švagr
4. května 2023

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především mému vedoucímu práce Ing. Pavlu Švagrovi za trpělivost, poznatky a cenné rady. Dále bych chtěl poděkovat své přítelkyni a rodině za podporu při psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 4. května 2023 .....

Abstrakt

Předmětem této práce je prozkoumání aktuálně používaných metod a nástrojů pro určení nálady v komunikačních aplikacích. Na základě tohoto průzkumu a praktických testů je vybrán model pro zjištění nálady a její predikci. Dále je vytvořena integrace této analýzy do komunikační aplikace Slack formou bota. Výsledky jsou srovnány s existujícím řešením. Nakonec je nástroj testován ve reálném firemním prostředí.

Klíčová slova nálada, analýza sentimentu, zpracování textu, strojové učení, časové řady, predikce, Slack, komunikační aplikace, Python

Abstract

The subject of this thesis is to investigate the currently used methods and tools for determining mood in communication applications. Based on this survey and practical tests, a model for mood detection and prediction is selected. Furthermore, an integration of this analysis into the Slack communication application in the form of a bot is developed. The results are compared with an existing solution. Finally, the tool is tested in a real corporate environment.

Keywords mood, sentiment analysis, text processing, machine learning, time series, prediction, Slack, communication applications, Python

Seznam zkratek

SA	Analýza sentimentu
OM	Opinion mining
NP	Noun phrases
TF-IDF	Term Frequency-Inverse Document Frequency
BoW	Bag of Words
MLE	Maximum likelihood estimation
SVM	Support Vector Machine
SVR	Support Vector Regression
SMOTE	Synthetic Minority Over-sampling TEchnique
ROC	Receiver operating characteristic
AUC	Area under the curve
NLTK	Natural Langue Toolkit
MAE	Mean Absolute Error

Kapitola 1

Úvod

Komunikační aplikace, jako jsou chatovací platformy a chatboti, jsou stále oblíbenějším prostředkem komunikace v osobním i profesním životě. S rozmachem strojového učení a umělé inteligence v posledních letech roste zájem o jejich využití. Analýza nálady může být jedním z nich. V komunikačních aplikacích buďto může mít za cíl zlepšit uživatelský zážitek, zjistit potřeby uživatele nebo mít čistě informační hodnotu. Analýza nálady je proces zjišťování a analýzy emočního stavu jednotlivce na základě jeho psaného nebo mluveného projevu.

Tato práce se zabývá hledáním vhodného modelu pro analýzu nálady a vytvořením vlastních metrik pro použití v komunikačních aplikacích. Po zjištění nálady budeme predikovat její budoucí hodnoty a zjišťovat trend. K tomu využijeme aparát časových řad. Tato analýza a predikce bude integrována do bota v aplikaci Slack a následně otestována na reálných datech z firemního prostředí. Tento nástroj by mohl být užitečný pro firmy a organizace, které chtějí sledovat morálku svých zaměstnanců nebo členů.

V první části bude čtenář seznámen s potřebným teoretickým základem. Budeme mluvit o metodách analýzy sentimentu, různých modelech strojového učení a nakonec o časových řadách, jejich vlastnostech a predikci. Dále pak ukážeme a popíšeme již existující řešení v různých programovacích jazycích, ale ukážeme si i samostatné nástroje a hotové integrace v některých komunikačních aplikacích. V další části pak ukážeme a vysvětlíme části implementace a budeme zde prezentovat výsledky konkrétních modelů. V poslední části si ukážeme výsledky z testování ve firemním prostředí.

1.1 Cíle práce

Cílem této práce je vytvořit nástroj na analýzu vývoje nálady na komunikačních platformách. Cílem teoretické části práce je analyzovat používané metody pro určení sentimentu. Dalším cílem je navrhnout vlastní metriky pro měření nálady. Cílem praktické části je vybrat vhodný model pro analýzu nálady a predikci vývoje nálady. Navržené řešení pak implementovat a vytvořit nástroj integrovaný do komunikační platformy Slack. Jeden z požadavků je poskytovat denní přehled vývoje nálady do určitých kanálů. Posledním cílem této části je otestovat řešení na reálných

datech z firemního prostředí.

Kapitola 2

Teoretická východiska práce

V této kapitole se budeme zabývat teoretickými východisky práce a vymezením používaných pojmů. Budeme se zde zabývat jak samotnou analýzou sentimentu tak rozebereme supervizované modely strojového učení. Nakonec si také popíšeme základní pojmy modelování časových řad a řekneme si něco o jejich analýze a predikci.

2.1 Analýza sentimentu

Na poli výzkumu dolování dat z textu můžeme narazit na dva pojmy, které jsou zaměnitelné. Jedná se o *Analýzu sentimentu (SA)* a *Opinion mining (OM)* [1]. Někteří autoři je ale rozlišují. Například Tsytsarau a Palpanas uvádějí [2], že OM extahuje a analyzuje názor lidí vázající se k nějakému objektu či entitě, kdežto SA pouze identifikuje sentiment vyjádřený v textu a ten pak analyzuje. Podle Medhata a spol. můžeme SA rozdělit na 3 klasifikační úrovně [1]:

- Document-level
- Sentence-level
- Aspect-based

Feldman k těmto úrovním přidává ještě další [3]:

- Komparativní

2.1.1 Document-level

Jedná se o nejjednodušší formu SA. Základní jednotkou informace této metody je jeden dokument a předpokládá se, že zde autor vyjadřuje svůj názor na jeden objekt. Existují dva hlavní přístupy řešení a to: supervizované a nesupervizované učení. Supervizované učení v tomto případě předpokládá, že existuje konečná množina tříd, reprezentující hledaný sentiment. V nejjednodušším případě si vystačíme s dvěmi třídami, konkrétně negativní a pozitivní. Obecně se tak jedná o nějakou diskrétní škálu hodnot. Nesupervizované učení využívá Semantic orientation (dále jen SO) konkrétních slov nebo vět.

2.1.2 Sentence-level

Pokud problém vyžaduje detailnější analýzu, než poskytuje Document-level, využijeme Sentence-level metodu. Prvotně musíme věty rozdělit na subjektivní, což je osobní nebo předpojatý pohled na věc, a objektivní, kde naopak se jedná o věcnou a skutečnou pravdu. Objektivní věty se pro další analýzu často nevyužívají, ale některé přístupy využívají oba typy vět. Po rozdělení můžeme opět využít buďto supervizované nebo nesupervizované učení, stejně tak jako u Dokument-level analýzy. Narayanan a spol. ukázali, že je vhodné na různé věty používat různé metody [4]. Konrétně věty různých typů: tázací, podmiňovací a sarkastické.

2.1.3 Aspect-level

Předchozí 2 typy byly předpokládaly že se každý dokument nebo věta váže k jednomu objektu. V mnoha případech však lidé hovoří o více objektech s více aspekty (objektem může být např. mobilní telefon, u kterého se můžeme vyjádřit k jeho aspektům (rychlost, velikost, paměť, atd.) kladně i záporně). Kdybychom analyzovali text o jednom objektu s více aspekty těmito přístupy, došlo by ke ztrátě mnoha informací. Aspect-level analýza se proto zaměřuje na jednotlivé aspekty objektu a k nim přiřazuje zjištěný sentiment.

Nejdříve tedy musíme identifikovat aspekty objektu. Většina komerčních produktů používá přístup, kde si nejdříve vyfiltrují všechny fráze s podstatnými jmény (dále jen NPs – noun phrases), kterých je určitý počet. [3] Dále pak můžeme zredukovat šum v NPs [3] pomocí PMI. Existují ale i další přístupy: Například phrase dependency parser, Conditional Random Field, atd. Tyto metody identifikují takzvané explicitní aspekty, ale v textu se mohou nacházet i implicitní aspekty (např. ve větě: „Toto auto je pomalé“ není nikde uveden aspekt rychlosti). „Finální polarita každého aspektu je determinována váženým průměrem polarit všech výrazů sentimentu nepřímo vážené vzdálenopsti mezi aspektem a výrazem sentimentu.“ [3, překlad vlastní]

2.1.4 Komparativní analýza

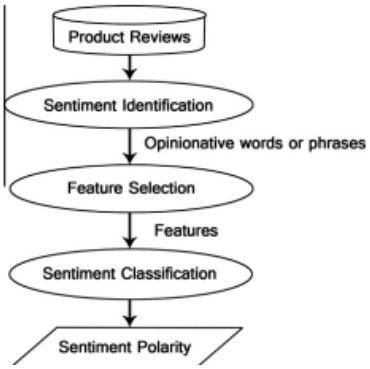
V některých případech se lidé nevyjadřují k objektech přímo, ale porovnávají 2 objekty mezi sebou (např. „Windows je lepší operační systém, než jakákoliv distribuce Linuxu“). Zde tedy musíme vyextahovat objekty z těchto vět. Autorům Jindal a Jiu se podařilo najít relativně malou množinu slov, která tyto věty dokáže identifikovat. [5] Jedná se o anglická slova, například o: [5]

- Srovnávací přídavná jména a příslovce – „more“ – více, „less“ - méně a slova končící koncovkou „-er“.
- Přídavná jména a příslovce v superlativu – „most“ – nejvíce, „least“ - nejméně a slova končící na „-est“.
- Jiné fráze – „than“ - než, „outperform“ – překonat, „up against“ - proti, ...

„Tyto klíčová slova a fráze (celkem 83) dokáží zachytit komparativní věty se senzitivitou 94%“. [5, překlad vlastní]

2.2 Metodika analýzy

Analýzu sentimentu si můžeme představit jako proces. Na obrázku 2.1 můžeme vidět její součásti. Data nejčastěji bývají z recenzí produktů, avšak to mohou být i novinové články nebo politické debaty. Ve fázi identifikace sentimentu vybíráme možné slova nebo fráze vyjadřující názor. Následuje výběr příznaků a klasifikace sentimentu, kde existuje více přístupů. [1]



Obrázek 2.1 Ukázka procesu sentimentové analýzy Zdroj: [1]

Tyto přístupy můžeme rozdělit na dvě základní skupiny: strojové učení a přístup založený na slovníku.

Metody strojového učení si popíšeme v následujících kapitolách. Druhý přístup spočívá ve vybudování slovníku slov nebo frází, které mají pozitivní, neutrální nebo negativní konotaci. Feldman rozlišuje manuální a automatické budování slovníku [3]. Manuální je o ručním ohodnocení jednotlivých slov a frází. Tento přístup není použitelný pro všechny problémy, ale můžeme získat kvalitnější a více přesné a specifické výsledky. Automatické využívá ke konstrukci slovníku strojové učení nebo model WordNet [3].

2.2.1 Strojové zpracování textu

K použití strojového učení k analýze sentimentu, budeme potřebovat předzpracovat data a extrahovat z nich příznaky. K extrakci se používají metody Bag of Words nebo TF-IDF (podrobněji v kapitolách 2.2.3 a 2.2.4). Následně na těchto příznacích natrénujeme různé modely s různými parametry a pomoci vhodných metrik tyto modely ohodnotíme. V následujících bodech/podsekcích si představíme základní pojmy, techniky čištění dat, techniky a důvody pro balancování dat, techniky k extrakci příznaků, modely a metriky.

V oboru zpracování textu se využívají tyto pojmy:

- Dokument (Document) – objekt obsahující text
- Kolekce (Collection) – množina dokumentů
- Slovo (Term) – jednotlivé slovo nebo fráze
- Slovník (Vocabulary) – množina všechn slov ze všech dokumentů v kolekci

- N-gram – posloupnost po sobě jdoucích n slov. Pro $n = 2$ je označován jako *bigram* a pro $n = 3$ jako *trigram*.

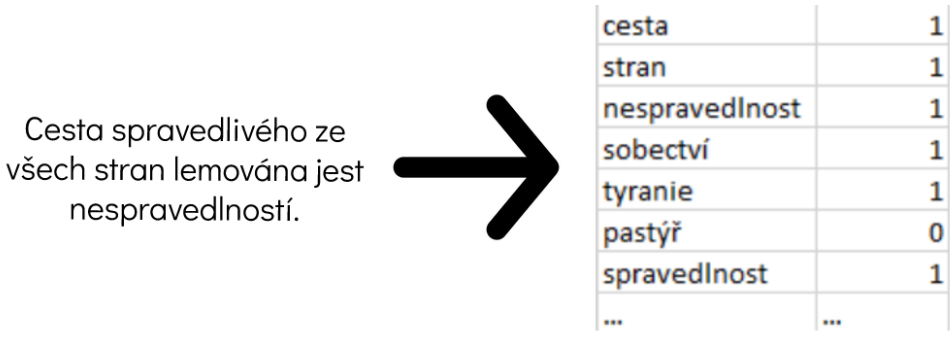
2.2.2 Stemming a lemmatizace

„Z gramatických důvodů, dokumenty využívají různé tvary slov (např. organizovat, organizuje, organizování) nebo existují příbuzná slova s podobným významem (např. demokracie, demokratický, demokratizace).“[6, překlad vlastní] V těchto situacích nechceme rozlišovat slova na základě jejich tvarů, ale na základě jejich významů. K tomu slouží techniky zvané **stemming** a **lemmatizace**. „Cílem obou techik, je redukovat tvary a někdy i příbuzná slova na společnou základnú formu.“[6, překlad vlastní] Například slova „je“, „jsem“ a „jsou“ se redukují na slovo „být“. Ačkoliv cíle obou technik jsou stejné, jejich principy jsou rozdílné. Stemming je heuristická metoda, která odstaňuje konce a začátky slov. Lemmatizace pak využívá slovníky a morfologickou analýzu slov.

„Nejrozšířenější algoritmus pro stemming anglických slov, u kterého se empiricky ukázala jeho vysoká efektivita, je *Porterův algoritmus*.“[6, překlad vlastní] Celý algoritmus je komplexní[1], ale jeho základní myšlenka je v postupném sekvenčním 5 fázovém redukování slov podle určitých pravidel. V každé fázi jsou jiné způsoby jak konkrétní pravidla vybírat. V pozdějších fázích algoritmu se bere v potaz i počet slabik, aby výsledek měl nějaký význam. Existují i jiné algoritmy pro stemming, ale jejich popis není v rozsahu této práce. Co se týče výsledků metrik (více v kapitole 2.2.7) „stemming zvyšuje sensitivitu a zároveň snižuje preciznost.“[6, překlad vlastní]

2.2.3 Bag of Words

Bag of Words je metoda, která z textu extrahuje příznaky pro možnost využití v modelech strojového učení [7]. Principem je prosté označení, jestli se určité slovo nachází v dokumentu či nikoliv. Tímto přijdeme o informaci o složení věty. Výsledkem metody je tedy převést dokument na vektor čísel (v primitivním případě můžeme uvažovat i pouze o binárních hodnotách tj. 0/1) označující počet výskytů slov.



Obrázek 2.2 Ukázka Bag of Words na příkladu.

[1] Detaily o algoritmu a implementace dostupné na tartarus.org/martin/PorterStemmer.

2.2.4 TF-IDF

Term Frequency-Inverse Document Frequency je statistická metoda, která měří míru důležitosti slova v dokumentech.[7] Vypočteme ji takto:

$$TF - IDF(d,t) = TF(t,d) * IDF(t),$$

kde t je slovo, d je dokument, $TF(t,d)$ je počet výskytů slova v dokumentu (viz. definice 2.1) a $IDF(t)$ je důležitost slova (viz. definice 2.2). Slovo je důležitější, pokud se nachází v méně dokumentech.

▶ **Definice 2.1** (Term Frequency)**.**

$$TF(t,d) = \frac{n_{t,d}}{\sum_k n_{k,d}},$$

kde $n_{t,d}$ je počet výskytů slova t v dokumentu d.

▶ **Definice 2.2** (Inverse Document Frequency)**.**

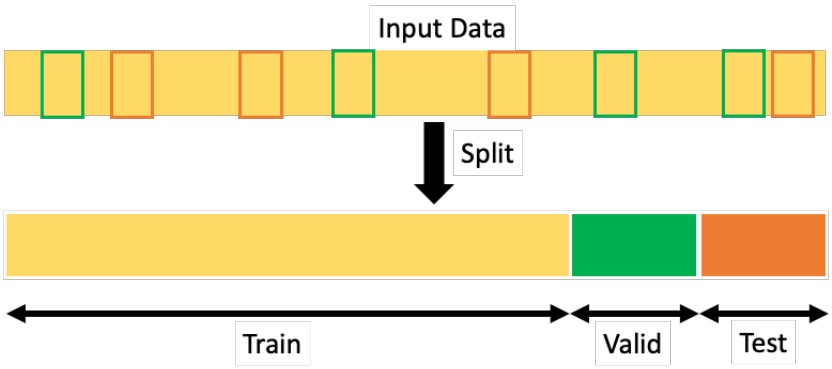
$$IDF(t) = log\frac{|D|}{|d : t \in d|},$$

kde $|D|$ je celkový počet dokumentů a jmenovatel výrazu je počet dokumentů obsahující slovo t.

2.2.5 Přístupy strojového učení

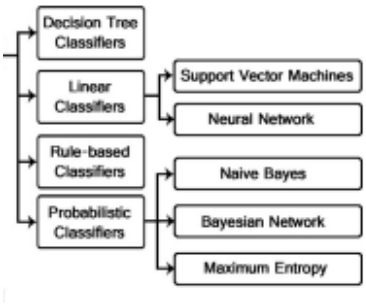
Strojové učení se běžně rozděluje na tyto skupiny: [8]

- Supervizované učení – Máme k dispozici vstupní data (příznaky) a k nim přiřazený správný výstup (ten může být diskrétní resp. spojitý a potom úlohu nazýváme klasifikační resp. regresní). Díky známým výstupům můžeme trénovat náš model, na základě chyb od originálního výstupu. Data se zde rozdělují do 3 skupin: trénovací, validační, testovací v určitém poměru (zde se zdroje neschodují na stejných hodnotách, většinou se ale jedná o poměr 3:1:1 viz. Obrázek 2.3). Trénovací slouží k trénování modelu a jeho parametrů, validační k jeho porovnání s ostatními natrénovanými modely a testovací pak k odhadu chyby našeho modelu.
- Nesupervizované učení – Zde nemáme ke vstupním datům odpovídající výstup a autor zde musí sám najít souvislosti.
- Semi-supervizované učení – Kombinace supervizovaného a nesupervizovaného učení
- Posilované učení – Ayodele jej definuje takto: „Algoritmus se učí pravidla, jak jednat při pozorování světa. Každá akce má nějaký dopad na prostředí a prostředí poskytuje zpětnou vazbu, která řídí algoritmus učení.“ [9, překlad vlastní]
- A další



Obrázek 2.3 Rozdělení dat pro učení supervizovaných modelů.

Medhat ve svém článku ukázal používáné metody zabývajících se analýzou sentimentu. [1] Výčet z kategorie strojového učení pak můžeme vidět na Obrázku 2.4. V následujících kapitolách si tyto metody/modely představíme.



Obrázek 2.4 Možnosti použití strojového učení k sentimentové analýze [1]

2.2.5.1 Logistická regrese

Nehledě na název, logistická regrese je model určený pro klasifikaci. Existují rožšíření tohoto modelu na multinomickou klasifikaci, ale pro náš případ, rozlišení pozitivní a neutrální nálady, bude stačit definice pouze pro binární klasifikaci. Svoje základy opírá o lineární regresi, která „hledá závislost v zadaném tvaru lineární kombinace příznaků

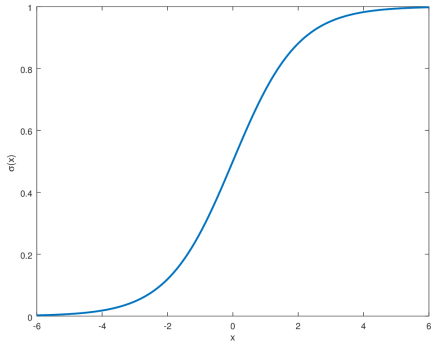
$$Y \approx w_0 + w_1 x_1 + \ldots + w_p x_p,$$

kde x_i jsou nějaké konkrétní hodnoty příznaků X_i a w_i jsou neznámé koeficienty.“[8] Konkrétně pomocí této lineární kombinace příznaků predikujeme pravděpodobnost $P(Y = 1)$ (konkrétněji $P(Y = 1|\mathbf{x}, \mathbf{w})$), tedy pravděpodobnost (podmíněnou hodnotou příznaků $\mathbf{x}$ a koeficientů $\mathbf{x}$), že vysvětlovaná proměnná Y má hodnotu (patří do třídy) 1.

Hodnotu lineárního výrazu, z lineární regrese, pak dosadíme do funkce, jejíž obor hodnot je podmožina intervalu $[0, 1]$. „Obvyklou volbou této funkce je sigmoida, což je speciální případ logistické funkce.“[8], viz. definice 2.3 a obrázek 2.5.

▶ **Definice 2.3** (Sigmoida). ***Sigmoida*** *je funkce s předpisem:*

$$f(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}.$$



Obrázek 2.5 Graf sigmoidy. Zdroj: hvidberrrg GitHub-Deep learning

Po dosazení bude model pro odhad pravděpodobnosti bude vypadat takto:

$$P(Y = 1|\mathbf{x}, \mathbf{w}) = \frac{e^{\mathbf{w}^\top \mathbf{x}}}{1+e^{\mathbf{w}^\top \mathbf{x}}}.$$

Pokud je hodnota výrazu větší než 0,5, predikce pro Y bude 1, jinak 0. [8]

Učení pak probíhá pomocí MLE odhadu. Pokud uvažujeme N trénovacích dat, skládajících se z vysvětlované proměnné Y_i a p příznaků

$$\mathbf{x}_i = (x_{i;0}, x_{i;1}, \ldots, x_{i;p}),$$

kde $i = 1, \ldots, N$, věrohodnostní funknce má tvar:

$$L(\mathbf{w}) = \prod_{i=1}^{N} P(Y = Y_i|\mathbf{x}_i, \mathbf{w}).$$

Tuto funkci chceme maximalizovat, kdy si můžeme pomoct zlogaritmováním věrohodnostní funkce (celý výpočet viz [10]):

$$\ln L(\mathbf{w}) = l(\mathbf{w}) = \sum_{i=1}^{N} (Y_i \mathbf{w}^\top \mathbf{x}_i - \ln(1 + e^{\mathbf{w}^\top \mathbf{x}_i})).$$

„Maximum bychom měli nalézt mezi řešeními rovnice $\nabla l(\mathbf{w}) = 0$, kdy na rozdíl od lineární regrese neumíme najít explicitní řešení a je tedy nutné použít numerické aproximativní metody (např. vícerozměrná verze Newtonovy metody, nebo gradientní vzestup).“ [8]

2.2.5.2 Metoda nejbližších sousedů

Metoda nejbližších sousedů (anglicky k-nearest neighbors algorithm nebo taky KNN nebo k-NN) je supervizovaná metoda používaná jak pro klasifikaci (diskrétní vysvětlovaná proměnná) tak pro regresi (spojitá vysvětlovaná proměnná). „Je třeba poznamenat, že algoritmus KNN je také součástí rodiny modelů "lazy learning", což znamená, že si pamatuje pouze trénovací datovou sadu oproti samotnému trénování modelu.“[11, překlad vlastní]

Výsledek predikce je založen na vzdálenosti predikovaného bodu od ostatních pomocí předem definovaných metrik, a na základě hodnot k (k je zde parametr) nejbližších sousedů je určen výsledek. Pro klasifikaci je výsledná třída určena pomocí „většinového hlasování“. [11] Při zvolení parametru $k = 1$ je výsledek určen pomocí nejbližšího souseda. Pro regresi se pak jako výsledek bere průměr všech k nejbližších sousedů. Metriky na výpočet vzdálenosti musí splňovat určité vlastnosti uvedené v následující definici [8]:

▶ **Definice 2.4.** ***Vzdálenost*** *nebo* ***metrika*** *na množině $\mathcal{X}$ je funkce $d : \mathcal{X} \times \mathcal{X} \to [0, +\inf)$ taková, že pro každé $x, y, z \in \mathcal{X}$ platí:*

- *Pozitivní definitnost* – $d(x, y) \geq 0$, *a* $d(x, y) = 0 \Leftrightarrow x = y$
- *Symetrie* – $d(x, y) = d(y, x)$
- *Trojúhelníková nerovnost* – $d(x, y) \leq d(x, z) + d(z, y)$

Existuje několik metrik, které zde můžeme použít, ale nejčastěji se využívá **Euklidovská vzdálenost**, která měří přímku mezi dvěma body. [11] Matematicky je definovaná takto:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

Další populární metrikou je **Manhattanská vzdálenost**:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Obě tyto metriky jsou konkrétní druhy tzv. Minkovského metrik s parametrem $p = 1, 2, \ldots$:

$$d_p(x, y) = \sqrt[p]{\sum_{i=0}^{n} |x_i - y_i|^p}.$$

S použitím těchto jednoduchých, leč účinných, metrik můžeme narazit na problém se špatným škálováním jednotlivých příznaků. Toho se můžeme zbavit jednoduchou normalizací

$$x_i \leftarrow \frac{x_i - min_x}{max_x - min_x}$$

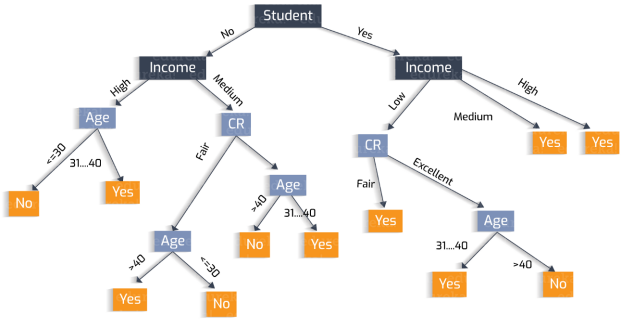
nebo standartizací

$$x_i \leftarrow \frac{x_i - \bar{x}}{sqrts_x^2},$$

kde $\bar{x}$ je výběrový průměr a s_x^2 je výběrový rozptyl. [8]

2.2.5.3 Rozhodovací stromy

Rozhodovací strom je jednoduchý model, který můžeme použít ke klasifikaci i regresi. Kotsiantis jej definuje takto: „Rozhodovací stromy jsou sekvenční modely, které postupně provádějí jednoduché testy, porovnání; každý test porovnává číselný atribut s prahovou hodnotou nebo s nominálním atributem soubor možných hodnot.“ [12, překlad vlastní] Russel a Norvig jej zase definují obecněji jako funkci, která jako vstup očekává vektor příznaků a vrací jednu hodnotu – rozhodnutí [13]. Oproti jiným modelům jsou rozhodovací stromy také lépe graficky reprezentované (pomocí stromové struktury viz obrázek 2.6) a rychlé k pochopení. Výsledek klasifikace resp. regrese k určitým příznakům zjistíme z daného listu, do kterého se vstupní data postupným porovnáváním dostala.



Obrázek 2.6 Schéma rozhodovacího stromu. Zdroj: Huawei Cloud blog

Než vysvětlíme jak stromy vytvářet, budeme k tomu potřebovat různé metriky, pomocí kterých určíme kvalitu rozdělení vektoru příznaků.

Jednou z metrik je tzv. **informační zisk** a vypočítáme ho jako[12]:

$$InformacniZisk(\mathcal{D}, X_i) = IG(\mathcal{D}) = H(\mathcal{D}) - \sum_{j=0}^{k-1} t_j H(\mathcal{D}_j),$$

kde $\mathcal{D}_j$ je podmnožina $\mathcal{D}$ pro které $X_i = j$, t_j je podíl počtu prvků v $\mathcal{D}_j$ a $\mathcal{D}$.

Funkce $H(\mathcal{D})$ ve vzorci je pak odhad entropie dat:

$$Entropie(\mathcal{D}) = H(\mathcal{D}) = -\sum_{i=0}^{k-1} p_i \log(p_i),$$

kde $\mathcal{D}$ je množina dat, p_i je poměr počtu i v $\mathcal{D}$ a platí $\sum_{i=0}^{k-1} p_i = 1$.

Další metrikou je tzv. **Gini index**, která udává míru špatné klasifikace nově přidaného prvku. Přesný výpočet je [8]:

$$GiniIndex(\mathcal{D}) = GI(\mathcal{D}) = \sum_{i=0}^{k-1} p_i(1 - p_i)$$

Gini index můžeme využit při výpočtu informačního zisku, kdy jen ve vzorci nahradíme $(\mathcal{D})$ za

$GI(\mathcal{D})$.

Narozdíl od parametrů, které hledáme při trénování (parametry = vrcholy, trénování = konstrukce stromu), má rozhodovací strom hyperparametry. Hyperparametry modelu jsou takové parametry, které se používají při konstrukci stromu. Určují používané metriky a hodnoty podmínek větvení, resp. vytváření listů. Hledání nejoptimálnějších hodnot se nazývá **ladění hyperparametrů** a jedná se o systematické zkoušení různých kombinací hodnot a vyhodnocování výsledků modelu. Mezi základní hyperparametry patří [14]:

-
- Kritérium – Funkce, která měří kvalitu rozdělení příznaků. Nejčastější jsou Gini index nebo Entropie. Tato funkce se pak použije při výpočtu informačního zisku.
- Maximální hloubka – Maximální hloubka rozhodovacího stromu. V každém úplném binárním stromě s hloubkou h je přesně 2^{h-1}. Kdybychom měli h příznaků [2], tak počet všech možnách kombinací je také 2^{h-1}, čímž by strom „degradoval“ na index. Díky tomuto hyperparametru můžeme této „degradaci“ předejít.
- Váha příznaků – Slouží k vážení důležitosti příznaku.
- Minimální počet výsledků v listu – Pokud při konstrukci stromu není dostatek příznaků, vytvoří se list.

K vytváření stromů pak existují algoritmy. Známý je ID3, který konstruuje stromy pomocí hladového přístupu. Jeho nevýhodou je předpoklad diskrétních dat a absence chybějích hodnot. Také se na malých datasetech přeučuje.

Algoritmus 1: Iterative Dichotomiser 3

```
Vstup: Matice příznaků, vektor vysvětlované proměnné
Výstup: Rozhodovací strom
if žádná data then
    return fail list
else if ∀ příklady mají stejný výsledek then
    return list, který vrací výsledek klasifikace
else if je splněna podmínka hyperparametru then
    return list, který vrací výsledek klasifikace
else
    A ← příznak nejlépe rozdělující vektor výsledků;
    r, l = split(A,X,y);
    levý_syn ← ID3(rX,rY);
    pravý_syn ← ID3(lX,lY);
end
```

Jeho vylepšení nástupci C4.5 a C5.0 už dokážou pracovat i se spojitými daty i chybějícími hodnotami. Od svého předchůdce se liší technikou **prořezávání**, při které po konstrukci stromu odstraňuje zbytečné větve. Tato technika nám pomůže proti přeučení stromu. Existují dva druhy

[2] Opět pro jednoduchost uvažujeme binární klasifikaci.

prořezávání, a to pre-prořezávání (při zjištění nespolehlivé informace přestaneme dále rozvíjet současnou větev) a post-prořezávání (nejdříve se zkonstruuje strom a zbytečné části se odstraní).

Posledním algoritmem je CART nebo-li Classification And Regression Trees a je velice podobný algoritmu C4.5. Narozdíl od něj ale umí pracovat se spojitou vysvětlovanou proměnou (tj. regrese). „CART konstruuje binární stromy pomocí prahových hodnot, které přinášejí největší zisk informací v každém uzlu.“ [12]

2.2.5.4 Naivní Bayesův klasifikátor

V rámci klasifikace můžeme narazit na speciální druh klasifikátorů, kterými jsou pravděbodobnostní klasifikátory. Klasické klasifikátory jsou reprezentované funkcí, která mapuje vstup (vektor příznaků) na výslednou třídu. Narozdíl od nich, pravděbodobnostní klasifikátory vrací podmíněnou pravděbodobnost $P(Y|X)$, kde Y je výsledná třída a X je vektor příznaků. Naivní Bayes spadá do kategorie pravděbodobnostních klasifikátorů.

▶ **Definice 2.5** (Podmíněná pravděbodobnost)**.**

$$P(Y|X) = \frac{P(Y,X)}{P(X)}$$

Existují dva přístupy, jak tuto pravděbodobnost vypočítat. [15] „Prvním je přímo vytvořit funkci, která vypočítá určité pravděbodobnosti $P(Y|X)$“ [15, překlad vlasntí]. Tento model pak nazýváme **diskriminativní**.

Druhým přístup nazýváme **generativní**. Pro každou hodnotu Y model naučíme $P(X|Y)$ a $P(Y)$. Poté budeme potřebovat Bayesovu větu [16]:

▶ **Definice 2.6** (Bayesova věta)**.** *Budiž $a_1, a_2, \ldots, a_n$ vzájemně disjunktní náhodné jevy, jejichž sjednocení tvoří celý pravděbodobnostní prostor a platí: $\forall i : P(A_i) > 0$. Pak pro libovný jev b takový, že $P(b) > 0$, platí:*

$$P(a_i|b) = \frac{P(a_i)P(b|a_i)}{P(b)} = \frac{P(a_i)P(b|a_i)}{\sum_{i=1}^{n} P(b_i|a)P(b_i)}$$

.

Po aplikování Bayesovy věty dostaneme chtěnou pravděbodobnost $P(Y|X)$ [15]:

$$P(Y|X) = \frac{P(X,Y)}{P(X)} = \frac{P(X|Y)p(Y)}{\sum_{Y'=1}^{C} P(X|Y')P(Y')}$$

. Označujeme ho generativním, protože představuje úplnou informaci o rozdělení, ze kterého byla data generována. Výslednou predikci klasifikační třídy pak určíme jako

$$\hat{Y} = arg\, max_y P(Y|X)$$

.

Klasifikátor nazýváme naivním, protože očekává nezávislost všech příznaků:

$$P(X|Y=c) = \prod_{i=1}^{D} P(X_i|Y=c)$$

. „I přes to, že toto obvykle neplatí (příznaky jsou většinou závislé), výsledný model je jednoduché natrénovat a funguje překvapivě dobře.“[15, překlad vlastní]

Rozlišujeme několik druhů, které odlišují především předpoklady, které činí ohledně rozdělení $P(x_i|y)$. [17] Nejznámější z nich pak jsou [17]:

- Gaussian Naive Bayes uvažuje gausovské rozdělení tj.:

$$P(x_i|y) = \frac{1}{2\pi\sigma_y^2} \exp - \frac{(x_i - \mu_y)^2}{2\sigma_y^2},$$

  kde σ_y a μ_y jsou parametry, které odhadneme pomocí metody MLE.

- Multinomial Naive Bayes (MNB) je používaný pro klasifikaci textů (kde data jsou typicky vektory počtů slov nebo případně TF-IDF vektory). Parametry distribuce jsou vektory $\Sigma_y = (\Sigma_{y1}, \ldots, \Sigma_{yn})$ pro každou třídu y, kde n je počet příznaků (v textové klasifikaci velikost slovníku) a Σ_{yi} jsou pravděpodobnosti $P(x_i|y)$. Tyto parametry teda označují pravděpodobnost, že se příznak i objeví ve vzorku patřícím do třídy y. Parametry Σ_{yi} se odhadují pomocí vyhlazené verze maximální věrohodnosti [17]:

$$\hat{\Sigma}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n},$$

  kde $N_{yi} = \sum_{x \in T} x_i$ je počet výskytů příznaku i v třídě y v trénovací množině a $N_y = \sum_{i=1}^{n} N_{yi}$ je počet všech příznaků v třídě y. Vyhlazovací parametr $\alpha >= 0$ zohledňuje možnost absence některých příznaků v trénovací množině a zabraňuje nulovým pravděbodobnostem.

- Complement Naive Bayes (CNB) je adaptace MNB, která je vhodná pro nevybalancované data. „Konkrétně CNB používá statistiky z doplňku každé třídy k výpočtu vah modelu. CNB dále často překonává MNB (často se značným náskokem) v úlohách klasifikace textu.“[17, překlad vlastní] Váhy a odhady parametrů se počítají takto:

$$\hat{\Sigma}_{ci} = \frac{\alpha_i + \sum_{j:y_j \neq c} d_{ij}}{\alpha + \sum_{j:y_j \neq c} \sum_k d_{kj}},$$

$$w_{ci} = log\hat{\Sigma}_{ci},$$

$$w_{ci} = \frac{w_{ci}}{\sum_j |w_{cj}|},$$

  kde $\alpha_i : \alpha = \sum_i \alpha_i$ jsou vyhlazovací hyperparametry, stejné jako u MNB, a d_{ij} v sumách je počet (nebo hodnota TF-IDF) slov i v dokumentu j, které nepatří do třídy c. Výsledek

klasifikace se pak určí podle nejmenší shody s doplňkem:

$$\hat{c} = \arg\min_{c} \sum_{i} t_i w_{ci}.$$

- Bernoulli Naive Bayes předpokládá, že data jsou rozdělena multivarietním Bernoulliho rozdělením.
- A další – Categorical Naive Bayes, Out-of-core Naive Bayes model fitting

2.2.5.5 SVM

Cílem SVM (Support Vector Machine) je v prostoru příznaků najít nadrovinu takovou, že rozdělí data do tříd s maximální možnou vzdáleností trénovacích bodů z rozdílné třídy od této nadroviny. Čím máme větší vzdálenost, tím menší je obecná chybovost modelu [18].

Pro nelineární případy se metoda zdá být výpočetně náročná, jelikož se jedná o lineární algoritmus v mnohadimenziobálním prostoru, díky využití jádrových funkcí a jádrovému triku [19][18].

▶ **Definice 2.7** (Bázové funkce). *Uvažujme prostor příznaků $\mathcal{X}$ a m lineárně nezávislých funkcí $\phi_1, \ldots, \phi_m$ takové, že $\phi_i : \mathcal{X} \to \mathbb{R}$ pro $\forall i = 0, \ldots, m$. Tyto funkce potom nazýváme bázové a představují transformace původních příznaků do nového m-rozměrného prostoru.*

▶ **Definice 2.8** (Jádrová funkce). *Funkci $k : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ nazveme jádrovou funkcí, pokud má předpis $k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{y})$ pro všechny $\boldsymbol{x}, \boldsymbol{y}$ z $\mathbb{R}^p$ a kde $\boldsymbol{\phi}(\boldsymbol{x})$ je vektor bázových funkcí.*

Metodu SVM můžeme využít jak pro klasifikaci, tak pro regresi. V našem případě (viz kapitola 4.2.1) si popíšeme pouze regresi. Uvažujme tedy, že máme n trénovacích vektorů $x_i \in \mathbb{R}^p$ a ke každému z nich odpovídající výstup z $\mathbb{R}$. SVR (Support Vector Regression) pak řeší úlohu minimalizace [18]:

$$\begin{aligned} \min_{w,b,\zeta,\zeta^*} \frac{1}{2} w^T w + C \sum_{i=1}^{n} (\zeta_i + \zeta_i^*) \\ \text{vzhledem k} \; & y_i - w^T \phi(x_i) - b \leq \varepsilon + \zeta_i, \\ & w^T \phi(x_i) + b - y_i \leq \varepsilon + \zeta_i^*, \\ & \zeta_i, \zeta_i^* \geq 0, i = 1, ..., n \end{aligned} \tag{2.1}$$

Predikce vzdálené alepsoň o ϵ od jejich skutečného cíle penalizujeme koeficienty ζ_i a ζ_i^*.

Pro velké datasety se hodí použít lineární verzi, kde problém minimalizace definujeme [18]:

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \max(0, |y_i - (w^T \phi(x_i) + b)| - \varepsilon),$$

kde ignorujeme chybu menší než ϵ.

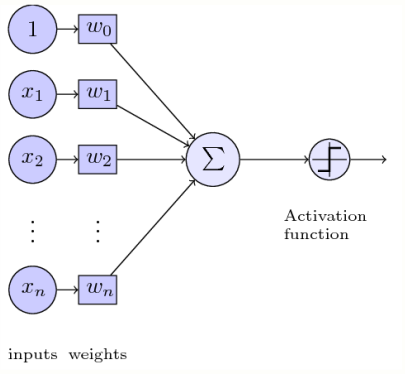
2.2.5.6 Neuronové sítě, percepton, vícevrstvý perceptron

Neuronové sítě se svojí strukturou inspirovaly strukturou vzájemně propojených neuronů v biologických systémech.

Základní stavební jednotkou je neuron, který:

- Přijímá reálné vstupy
- Produkuje jeden výstup

Nejjednodušším modelem neuronové sítě určeným ke klasifikaci je tzv. jednovrstvý perceptron [8] (schéma viz. 2.7).



Obrázek 2.7 Schéma perceptronu

K získání výstupu perceptronu potřebujeme znát jeho vnitřní potenciál y, který je váženým součtem vstupů.

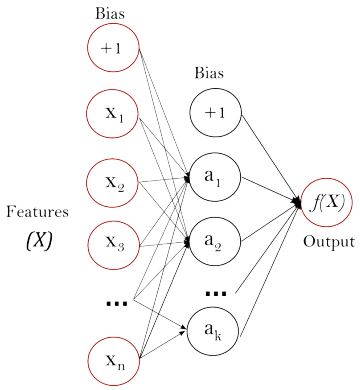
$$y = w_0 + \sum_{i=1}^{N} w_i x_i,$$

kde w_0 nazýváme intercept (anglicky bias)[20, překlad vlastní]. Tento potenciál je vstupem tzv. aktivační funkce. V případě perceptronu uvažujeme jako aktivační funkci tzv. skokovou funkci dánou předpisem [20, překlad vlastní]:

$$f(y) = \begin{cases} 1 & \text{když } \xi \geq 0, \\ 0 & \text{když} \xi < 0 \end{cases}.$$

Vícevrstvý perceptron (MLP) je algoritmus supervizovaného učení, který se snaží aproximovat funkci $f : R^n \to R^m$ trénováním na datasetu, kde n je počet dimenzí pro vstup a m je počet dimenzí pro výstup [21]. Narozdíl od jednoduchého perceptronu, dokáže rozlišit data, která nejsou lineárně separovatelná. Skládá se z minimálně tří vrstev: vstupní, výstupní a jedné nebo více skrytých vrstev.

Při pohledu na obrázek 2.8 můžeme zleva vidět vstupní vrstvu, která reprezentuje příznaky. Poté následují skryté vrstvy (na obrázku pouze 1), které opět, jako u perceptronu, počítají váženou sumu příchozích hodnot, kterou pak dosazují do aktivační funkce. Nakonec následuje



Obrázek 2.8 Vícevrstvý perceptron. Zdroj: [21]

poslední výstupní vrstva, která „přijímá hodnoty z poslední skryté vrstvy a transformuje je na výstupní hodnoty.“[21, překlad vlastní]

K učení se zde využívá tzv. backpropagation[3]. MLP lze použít jako klasifikátor i regresor. Při regresi se pouze nepoužívá aktivační funkce ve výstupní vrstvě. Výhodou MLP je schopnost chovat se jako nelineární model a učit se v reálném čase (online) [21]. Jeho nevýhody pak jsou [21]:

- Mnoho hyperparametrů pro ladění
- Citlivost na škálování příznaků
- Při trénování se používá ztrátová funkce, která se minimalizuje, a ta může mít několik lokálních minim.

2.2.6 Balancování dat

Nerovnováha tříd v úloze klasifikace je běžným problémem v několika oblastech. Zvláště pak v lékařství nebo při detekci podvodů. „Detekce vzácných onemocnění je typickým příkladem, kdy je přítomno mnoho vzorků bez dané nemoci, zatímco existuje jen velmi málo vzorků, které by reprezentovaly dané onemocnění.“ [22, překlad vlastní]

Balancovánání rozlišujeme na data level a algorithmic level přístupy. Data level přístup se využívá v předzpracování dat a existují různé metody, které si hned následně popíšeme. Algorithmic level spočívá v designu takových modelů, které si s nevybalancovanými daty poradí sami.

Pokud máme dostatek dat, můžeme použít **undersampling** (podvzorkování), kde se *eliminují* data ze třídy s větším zastoupením. Pokud nemáme dostatečný počet dat, využijeme **oversampling** (převzorkování), při kterém se duplikují nebo vytváří nové data z minoritních tříd. Metody oversamplingu [22]:

- Random oversampling – Při použití této metody vybíráme náhodné data z minoritních tříd a ty pouze zkopírujeme. Jelikož s kopírovanými daty nic jiného neděláme, variabilita celého

[3]Více viz. https://ieeexplore.ieee.org/document/118638

datasetu se vůbec nezmění a kvůli tomu hrozí přeučení modelu.

- SMOTE (Synthetic Minority Over-sampling TEchnique) – Tato metoda opět vybere data z minoritních tříd, najde k nejbližších sousedů a vytvoří nový bod, *posunutý* ve směru k jeho nejbližším sousedům (ke všem nebo jen k nějakým). Tato metoda tedy pouze nekopíruje, ale přidává varianci, která je ale kontrolovaná pomocí sousedů.

- A jiné

Metody undersamplingu [23]:

- Random undersampling – Podobný princip jako u oversamplingu. Vybíráme náhodné data z majoritních tříd a ty odstraníme. Jelikož k vybíráme náhodně bez žádných heuristik, můžeme přijít o důležitá data.

- Condensed nearest neighbors (CNN) – Data z majoritní třídy se odeberou pokud jednotlivé případy nejdou korektně klasifikovat. Přesněji pak pro balancování se do finálního datasetu berou všechny data z minoritní třídy a z majoritní třídy se vyfiltrují ty, jak už bylo řečeno, které nejdou správně klasifikovat.

- TomekLinks – Jedná se o modifikace CNN, která odtraňuje její nevýhody náhodnosti. Hledá zde určité páry bodů, které splňují určité podmínky „Body a a b jsou nazývané jako Tomek Link, pokud: nejbližším sousedem bodu a je b, nejbližším sousedem bodu b je a a body a a b patří do různých tříd.“[23, překlad vlastní] Tyto body jsou pak důležité při klasifikaci, tudíž je ponecháme v datasetu.

- A jiné

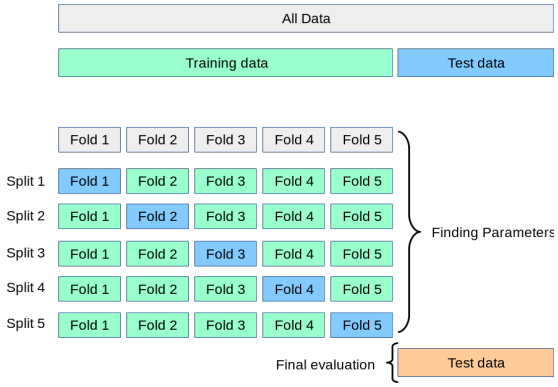
Při evaluaci modelů na nevybalancovaných datech se nemůžeme spolehnout na jednu kontrétní metriku, která by reprezentovala výsledky. Více informací o jednotlivých metrikách uvedeme v následující kapitole.

2.2.7 Metriky a evaluace modelů

V podkapitole 2.2.5 a konkrétně na obrázku 2.3 jsme si ukázali, jak se k hledání hyperparametrů a trénování modelu používají rozdělená data na trénovací, validační a testovací množiny. K eliminaci prvku náhody můžeme využít metodu zvanou **křížová validace**. Data se na počátku rozdělí pouze na dvě skupiny a to trénovací a testovací. Testovací množina bude použita na konci procesu k finální evaulaci modelu. V základním přístupu, který se nazývá **k-fold křížová validace**, se trénovací množina rozdělí na k menších množin. Poté pro každou menší množinu se jedna použije jako validační množina a zbylých $k-1$ malých množin se použije jako trénovací množina. Měřítko výkonnosti modelu je pak průměr hodnot metrik z každého *validování*.

Konkrétní metriky si nyní představíme.

Vyhodnocování klasifikačních modelů provádí pomocí měr odvozených z tzv. matice záměn (anglicky confusion matrix), což je matice četností různých predikovaných hodnot $\cap Y_i$ proti



Obrázek 2.9 Ukázka křížové validace Zdroj: [24]

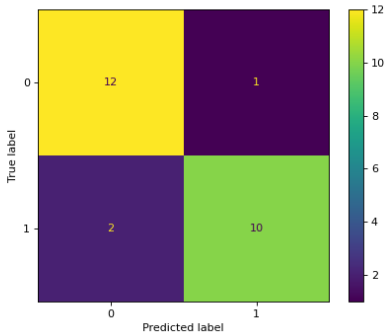
různým skutečným hodnotám Y_i [8]. Ukážeme si ji pro binární klasifikaci, ale můžeme ji použít i pro klasifikaci do více tříd. Má rozměr 2×2 a tvar

$$\begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix},$$

kde každý prvek je definován takto [8]:

- TP – True positive – počet správně klasifikovaných dat, které patří do třídy $Y = 1$.
- TN – True negative – počet správně klasifikovaných dat, které patří do třídy $Y = 0$.
- FP – False positive – počet špatně klasi v kolik dat které patří do třídy $Y = 1$.
- FN – True positive – počet špatně klasifikovaných dat které patří do třídy $Y = 0$.

Navíc si ještě zavedeme sloupcové součty N_+ resp. N_-, které značí počty bodů ve třídě 1 resp. 0.



Obrázek 2.10 Matice záměn (confusion matrix) Zdroj: [24]

Dále z této matice můžeme odvodit i jiné míry, konkrétně pak [8]:

- True positive rate – $TPR = \frac{TP}{N_+}$, také označováno jako *sensitivita* nebo *recall*.

- True negative rate – $TNR = \frac{TN}{N_-}$, také označováno jako *specificita*.
- False positive rate – $FPR = \frac{FP}{N_-}$, také označováno jako *chyba 1. druhu*.
- False negative rate – $FNR = \frac{FN}{N_+}$, také označováno jako *chyba 2. druhu*.
- Positive predictive value – Také označována jako *preciznost*. $PPV = \frac{TP}{\cap N_+}$, kde $\cap N_+$ označuje počet dat predikované modelem jako patřící do třídy 1.

Hojně používanou metrikou je **přesnost** (anglicky **accuracy**), vycházející z matice záměn, která je pro binární klasifikaci definovaná

$$ACC = \frac{TP + TN}{N},$$

a obecně ji můžeme definovat takto:

$$ACC = \frac{1}{N}\sum_{i=1}^{N} 1(\hat{Y}_i = Y_i),$$

kde $1(x)$ značí indikační funkci [24]. Přesnost není vhodná pro nevybalancované datasety, protože model označující všechny testovací data jako majoritní třídu by měl vysokou přesnost, ale ve skutečnosti by nebyl vůbec k použití.

Přesnost je speciální případ generalizované metriky zvané **top-k přesnost**, která predikci považuje za úspěšnou, pokud skutečná třída je alespoň v nejlepších k odhadech.

$$\texttt{top-k ACC} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{k} 1(\hat{f}_{i,j} = Y_i),$$

kde $\hat{f}_{i,j}$ je j-tá největší pravděbodnobnost, že se jedná o třídu i.

Další používanou metrikou je F_1**-skóre**. Jedná se o harmonický průměr *preciznosti* a *sensitivity* [8]

$$F_1 = \frac{2}{\frac{1}{PPV} + \frac{1}{TPR}} = 2\frac{PPV \cdot TPR}{PPV + TPR}.$$

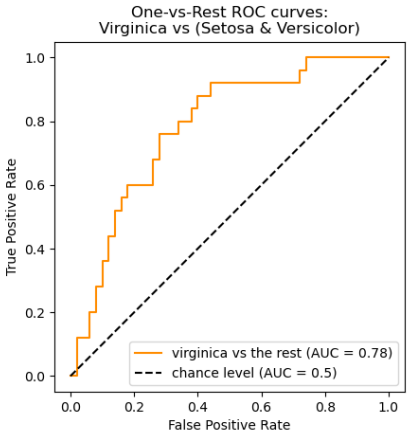
F1-skóre se používá při predikci na nevybalancovaných datasetech. Opět existuje generalizovaná verze, F_β**-skóre**, která pomocí parametru β zvětšuje důležitost senzitivity na úkor preciznosti:

$$F_\beta = (1 + \beta^2)\frac{PPV \cdot TPR}{(\beta^2 \cdot PPV) + TPR}.$$

F_β-skóre se pohybuje v intervalu od 0 do 1, kde 1 je nejlepší skóre a 0 nejhorší (to stejné platí o F_1-skóre, které je speciální případ F_β-skóre, když $\beta = 1$).

Další užitečnou metrikou může být **ROC-AUC** neboli *plocha pod ROC křivkou* (anglicky area under the ROC curve). Nejdříve si tedy ukážeme, co je **ROC křivka**.

ROC křivka jde získat pouze pro pravděbodobnostní klasifikátory, které neklasifikují data přímo, ale odhadují pravděpodovnost $P(Y = 1)$, že se jedná o třídu 1. Jak vidíme na obrázku 2.11, ROC křivka vzniká z *TPR* a *FPR*, kdy postupně měníme prahovou hodnotu, potřebnou



Obrázek 2.11 ROC křivka Zdroj: [24]

pro klasifikaci na základě pravděbodobnosti. Tím nám vznikne křivka se schodovou strukturou. Vypočtením velikosti plochy mezi křivkou a osou x získáme chtěnou metriku **ROC-AUC**. Tato metrika se opět pohybuje v intervalu od 0 do 1.

Pokud $ROC - AUC = 0$ tak model všechny data patřící do třídy 1 klasifikoval jako data patřící do třídy 0, nebo všechny data patřící do třídy 0 klasifikoval jako data patřící do třídy 1 (tj. pro všechny prahové hodnoty $TPR = 0$ nebo$FPR = 1$).

Pokud naopak $ROC - AUC = 1$ tak model všechny data patřící do třídy 1 klasifikoval správně jako data patřící do třídy 1, nebo všechny data patřící do třídy 0 klasifikoval jako data patřící do třídy 0.

Z toho vyplívá, že čím větší ROC-AUC, tím lepé umí model použít predikované pravděpodobnosti k rozlišení tříd.

2.3 Časové řady

Časové řady se vždy používaly na poli ekonometrie, kde se objevily na konci 30. let minulého století, konkrétně v prvním ekonometronomickém modelu pro USA od Jana Tinbergena [25]. V té době však nikdo neuvažoval, že by chronologicky seřazené pozorování mohly mít vliv na ostatní, ale uvažovala se nezávislost jednotlivých pozorování. Během následujících let se problémům tohoto tvrzení o nezávislosti v čase začlo zabývat více výzkumníků, ale největší pokrok nastal v 70. letech při publikaci od George E. P. Boxe a Gwilyma M. Jenkisne o analýze časových řad [25]. V současnosti se časové řady a forecasting používají v mnoha oborech např. v plánování elektráren, meteorologii, obchodování, marketingu, atd.

Časové řady se nejčastěji využívají na předpověď/predikce/forecasting (dále už jen jako predikce), ale často se tato úloha zaměňuje s plánováním a cíli.

- **Predikce** jsou co nejpřesnější předpovídání budoucího stavu na základě historických dat. Rozdělují se podle doby předpovědi na krátkodobé, střednědobé a dlouhodobé a každá kategorie má jiné využití (např. krátkodobé se využívají v plánování produkce nebo dopravy, ale

dlouhobobé se využivají ve strategickém plánování či demografii).

- **Cíle** jsou plánované žádoucí výsledky. Měly by být propojeny s predikcemi a plány, ale často se tak neděje.
- **Plánování** jsou reakce na predikce a cíle. Zahrnuje určování akcí potřebných ke schodě predikcí a cílů.

[26]

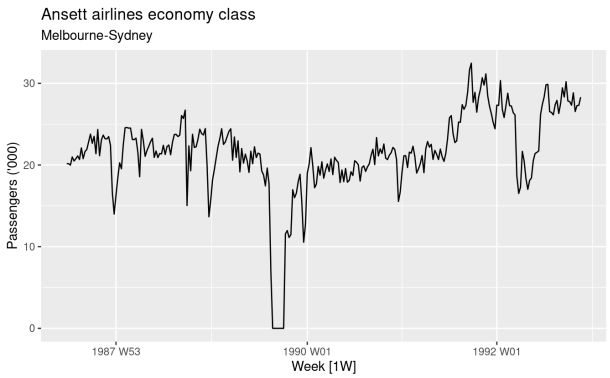
2.3.1 Definice a pojmy

Časovou řadu Hamilton definuje takto:

▶ **Definice 2.9.** *„**Časová řada** je definována jako soubor kvantitativních pozorování uspořádaných v chronologickém pořadí. Obecně předpokládáme, že čas je diskrétní proměnná. “[27, překlad vlastní]*

Formálněji pak časovou řadu můžeme definovat:

▶ **Definice 2.10.** *„Buď (Ω, Φ, P) pravděpodobnostní prostor a T množina indexů interpretovaných jako čas. **Časovou řadou** nazýváme množinu $(X_t, t \in T)$, kde X_t jsou náhodné veličiny z (Ω, Φ, P). “[25]*



Obrázek 2.12 Ukázka grafu časové řady Zdroj: [26]

Grafické zobrazení časové řady můžeme vidět na obrázku 2.12. Dále pak Pokud jsou pak indexy času diskrétní resp. spojité, mluvíme pak o časové řadě s diskrétním resp. spojitým časem.

„Řadu náhodných veličin $X_t : t = 0, 1, 2, \ldots$ pak nazýváme stochastickým procesem a používá se jako model pro pozorovanou časovou řadu.“ [28] Celý proces je pak určen množinou rozdělení všech konečných souborů Y, s čím nemusíme explicitně pracovat, protože spousta informací těchto sdružených distribucí jde popsat pomocí středních hodnot, variancí a kovariancí.

▶ **Definice 2.11.** *Pro stochastický proces $X_t : t = 0, 1, 2, \ldots$ definujeme **střední hodnotu** jako:*

$$\mu_t = E(Y_t) \text{ pro } t = 0, 1, 2, \ldots.$$

▶ **Definice 2.12.** *Pro stochastický proces* $X_t : t = 0, 1, 2, \ldots$ *definujeme* ***varianci*** *jako:*

$$\sigma_t^2 = var(Y_t) \text{ pro } t = 0, 1, 2, \ldots.$$

Namísto kovariance zde zavádíme pojem autokovariance, který je definován takto:

▶ **Definice 2.13.** *Pro stochastický proces* $X_t : t = 0, 1, 2, \ldots$ *definujeme* ***autokovarianci*** *jako:*

$$\gamma_{t,s} = cov(Y_t, Y_s) = E[(Y_t - \mu_t)(Y_s - \mu_s)] \text{ pro } t, s = 0, 1, 2, \ldots.$$

Nakonec můžeme definovat autokorelaci (ACF):

▶ **Definice 2.14.** *Pro stochastický proces* $X_t : t = 0, 1, 2, \ldots$ *definujeme* ***autokorelaci*** *jako:*

$$\rho_{t,s} = corr(Y_t, Y_s) \text{ pro } t, s = 0, 1, 2, \ldots,$$

kde

$$corr(Y_t, Y_s) = \frac{cov(Y_t, Y_s)}{\sqrt{var(Y_t)var(Y_s)}} = \frac{\gamma_{t,s}}{\sqrt{\gamma_{t,t}\gamma_{s,s}}}.$$

Hodnoty $\rho_{t,s}$ blízké ± 1 značí silnou závislost, avšak pouze lineární. Pokud $\rho_{t,s}$ se rovná 0, můžeme říct, že Y_t a Y_s jsou nekorelované [28].

Klasická korelace tedy značí lineární závislost mezi dvěmi veličinami, autokorelace nám měří lineárni závislost dvou zpožděných hodnot časové řady.

K získání dalších informací můžeme využít parciální autokorelaci (PACF), která odstraňuje lineární vliv mezilehlých hodnot mezi dvěmi okamžiky, a pak vypočítá autokorelaci.

Tyto dvě funkce (ACF, PACF) můžeme využít při odhadu řádů modelů.

Při výběru vhodného modelu pak můžeme využít různá kritéria. Jedním z nich je final prediction error (FPE) kritérium, kde řády modelu volíme podle predikcí modelu s nejmenší MSE (mean square error). Další z nich je Akaikeho informační kritérium (AIC), které se vypočítá [29]:

$$AIC = -2\ln(\text{maximální hodnota věrohodnosti}) + 2(\text{počet odhadovaných parametrů}).$$

Akaike toto kritérium modifikoval a vytvořil Bayesovské informační kritérium (BIC), kde výraz 2(počet odhadovaných parametrů) v AIC nahradíme výrazem $(p + p \log N)$, kde p je počet odhadovaných parametrů a N je počet pozorování, na kterých je model trénován [29].

Příklady

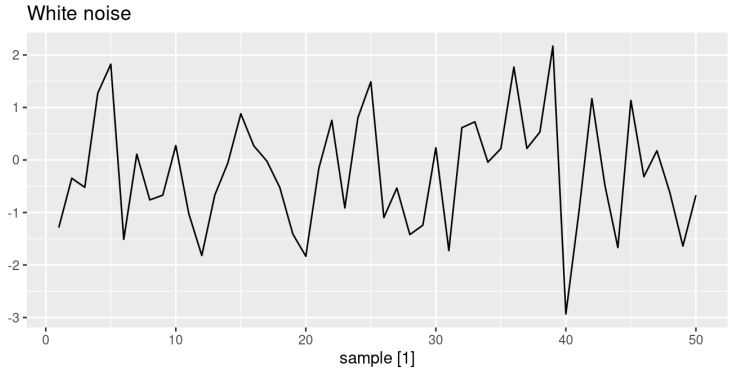
Jako známý příklad procesu můžeme ukázat **bílý šum** (viz obrázek 2.13). Je to proces Y_t, který splňuje:

$$\mathbb{E}[Y_t] = 0,$$

$$var(Y_t) = \sigma^2 < \inf,$$

$$cov(Y_t, Y_{t+\tau}) = \gamma(\tau) = 0,$$

Tedy je to proces, který má nulovou střední hodnotu, konstantní konečnou varianci a s nekorelovanost mezi Y_t a $Y_{t+\tau}$ pro $\tau > 0$. Jednotlivá Y_t jsou nezávislá a identicky rozdělená [27].



Obrázek 2.13 Bílý šum Zdroj: [26]

Jako další příklad můžeme ukázat **náhodnou procházku**. Uvažujme bílý šum X_t. Pak proces Y_t nazveme **náhodnou procházkou** jestli:

$$Y_0 = 0 Y_t = Y_{t-1} + X_t$$

a tedy

$$Y_t = \sum_{i=1}^{t} X_t.$$

Postupně tedy nasčítáváme bílé šumy a můžeme zjistit, že $\mathbb{E}[Y_t] = 0$ a $var(Y_t) = t\sigma^2$ „Náhodná procházka je oblíbeným modelem pro modelování cen akcií, ve fyzice pro popis Brownova pohybu, v computer science pro odhad velikosti webu atd.“ [27]

2.3.2 Stacionarita

Pro zkoumání struktury procesů nebo časových řad často musíme (racionálně) zjednodušovat některé předpoklady ohledně jejich struktur. „Nejdůležitější z nich je předpoklad stacionarity. Základní myšlenka stacionarity spočívá v tom, že pravděpodobnost které řídí chování procesu, se v čase nemění.“ [28, překlad vlastní]

Formálně pak:

▶ **Definice 2.15.** *Proces Y_t je **striktně stacionární** jestli sdružená distribuce $Y_{t_1}, Y_{t_2}, \ldots, Y_{t_n}$ je stejná jako sdružená distribuce $Y_{t_1-k}, Y_{t_2-k}, \ldots, Y_{t_n-k}$ pro všechny časové okamžiky $t_1, t_2, \ldots, t_n$ a všechny časové zpoždění k*

Vlastnost striktní stacionarity je velmi silná a proto rozlišujeme ještě **slabou stacionaritu**.

▶ **Definice 2.16.** *Proces Y_t je **slabě stacionární** pokud platí:*

- *Střední hodnota je konstantní v průběhu času, a*
- *$\gamma_{t,t-k} = \gamma_{0,k}$ pro všechny časové okamžiky t a zpoždění k*

Ke změně **ne**stacionární časové řady na stacionární nám může (ale nemusí) pomoct **diferencování**. Prakticky z řady Y_t vytváříme řadu $Y_t^{'}$ následovně:

$$Y_t^{'} = Y_t - Y_{t-1}, t = 1, 2, \ldots$$

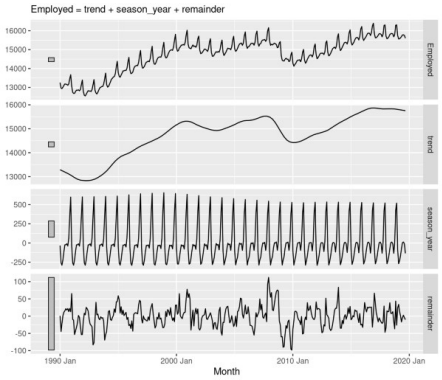
[27]

2.3.3 Dekompozice časové řady

Časovou řadu jde rozdělit na několik složek [26]:

- Trend – Existuje, pokud v časové řadě je dlouhodobý nárust nebo klesání hodnoty. Přesněji se jedná o dlouhodobý růst stední hodnoty.
- Sezónní složka – Jedná se o pravidelný, opakovaný jev. Může být ovlivněna například dnem v týdnu nebo měsíci v roce. Sezónost můžeme zjistit, po odstranění trendu, pomocí autokorelační funkce.
- Cyklické změny – Změny bez pevné periody, narozdíl od sezónní složky.
- Vše ostatní – Všechny ostatní jevy, co nepatří do ostatních složek. Jedná se většinou o nepravidelné fluktuace.

„Při rozkladu časové řady na složky obvykle spojujeme trend a cyklické změny do jediné složky (často se pro zjednodušení nazývá pouze trend). Časovou řadu si tedy můžeme představit jako složenou ze tří složek: trend, sezónní složka a zbytková složka (obsahující cokoli jiného v časové řadě).“ [26, překlad vlastní]



Obrázek 2.14 Ukázka dekompozice časové řady zaměstnanosti v maloobchodech USA. Zdroj: [26]

Při rozkladu využíváme dva modely a to **aditivní** a **multiplikativní**. Aditivní model je definován:

$$y_t = T_t + S_t + R - t,$$

kde y_t jsou data, T_t trend, S_t sezónní složka a R_t zbytek, vše v čase t. Multiplikativní model je pak definován následovně:

$$y_t = T_t \times S_t \times R - t.$$

„V aditivních modelech obecně platí, že amplituda sezónních složek je přibližně stejná, zatímco v multiplikativních se s rostoucím trendem zvyšuje i sezónní amplituda (a naopak).“[27] Alternativně můžeme data nejdříve vhodně transformovat a pak využít aditivní model. Lze totiž využít následující vztah:

$$y_t = T_t \times S_t \times R - t \Leftrightarrow \log y_t = \log T_t + \log S_t + \log R - t.$$

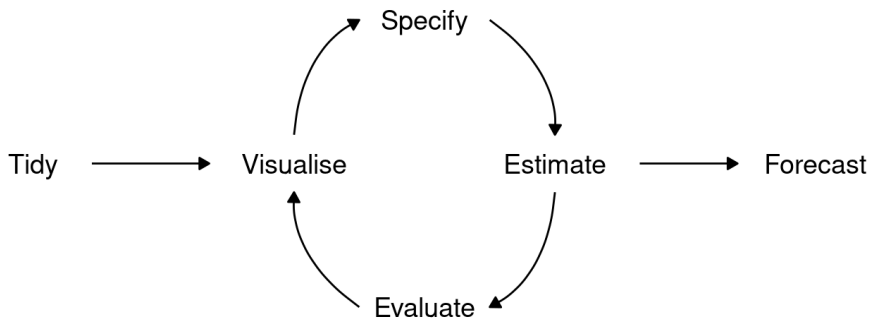
[26]

2.3.4 Predikce

Podle Hyndmana a Athanasopulose je predikování proces používání statistik a modelů k předpovídání budoucí hodnoty „nějaké veličiny“ na základě předchozích dat. [26] Zahrnuje vytváření modelů na základě hisotrické analýzy a jejich použití k pozorování. Hyndman a Athanasopulos také představili proces pro forecasting, který lze rozdělit na několik kroků (viz obrázek 2.15). Tento proces se skláda z [26]:

- Čištění dat – Indexacedat časem nebo nahrazování chybějících hodnot.
- Vizualizace – Pomůže nám identikovat vzorce v datech nebo odhadnout typ a parametry modelů.
- Definování modelu – Určení druhu modelu a jeho parametrů.
- Odhad – Trénování modelu na dostupných datech.
- Evaluace – Zjišťování přesnosti a výkonu modelu.
- Predikce – Finální krok, predikce budoucích hodnot na určenou dobu.

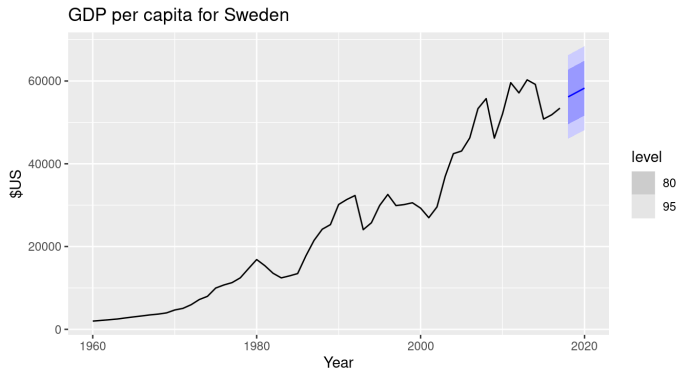
  [26]



Obrázek 2.15 Proces forecastingu. Zdroj: [26]

Výsledek takového procesu můžeme vidět na obrázku 2.16, kde jsou vidět historická data a predikci do budoucích let spolu s konfidenčními intervaly.

V následujících podkapitolách si představíme nejznámější modely hojně využívané v praxi.



Obrázek 2.16 Forecasting HDP na obyvatele Švédska Zdroj: [26]

2.3.4.1 Autoregresní modely

„V autoregresním modelu předpovídáme proměnnou, která nás zajímá, pomocí lineární kombinace minulých hodnot této proměnné. Termín autoregrese označuje, že se jedná o regresi proměnné proti sobě samé.“[26, překlad vlastní] Jinak řečeno současnou hodnotu proměnné získáme lineární kombinací p posledních hodnot z minulosti plus náhodné veličině reprezentující šum. Číslo p pak označujeme jako řád modelu. Formálně pak:

▶ **Definice 2.17.** ***Autoregresní model*** *p-tého řádu, zkráceně $AR(p)$, je definován jako*

$$Y_t = c + \Phi_1 Y_{t-1} + \Phi_2 Y_{t-2} + \ldots + \Phi_p Y_{t-p} + \epsilon_t,$$

kde $[,\Phi_1, \ldots, \Phi_p]$ je vektor regresních koeficientů a ϵ_t je bílý šum

Speciálně pak AR(1) model (tj. $Y_t = c + \Phi_1 Y_{t-1}$):

- kde $\Phi_1 = 0$ a $c = 0$ je bílý šum,
- kde $\Phi_1 = 1$ a $c = 0$ je náhodná procházka,
- kde $\Phi_1 = 1$ a $c \neq 0$ je náhodná procházka s driftem,
- kde $\Phi_1 < 0$ pak Y_t osciluje kolem střední hodnoty.

Pro zachování stacionarity jsou pak nutné omezující podmínky pro regresní koeficienty, konkétně pak:

- Pro AR(1) model: $-1 < \Phi_1 < 1$.
- Pro AR(2) model: $-1 < \Phi_2 < 1, \Phi_1 + \Phi_2 < 1$ a $\Phi_2 - \Phi_1 < 1$.

2.3.4.2 Modely klouzavých průměrů

Místo historických hodnot, jako v AR modelech, můžeme časovou řadu modelovat pomocí chyb/šumu z minulosti.

▶ **Definice 2.18.** ***Model klouzavých průměrů*** *q-tého řádu, zkráceně MA(q), je definován jako*

$$Y_t = c + \Theta_1\epsilon_{t-1} + \Theta_2\epsilon_{t-2} + \ldots + \Theta_q\epsilon_{t-q} + \epsilon_t$$

, kde $[c, \Theta_1, \ldots, \Theta_q]$ *je vektor regresních koeficientů a* ϵ_t *je bílý šum*

Pro zachování stacionarity jsou pak nutné omezující podmínky pro parametry Θ, konkétně pak:

- Pro MA(1) model: $-1 < \Theta_1 < 1$.
- Pro MA(2) model: $-1 < \Theta_2 < 1$,$\Theta_1 + \Theta_2 > -1$ a $\Theta_1 - \Theta_2 < 1$.
- Více komplikované pro $q >= 3$

[26]

V některých případech můžeme určené MA modely přepsat na AR modely a naopak. Každý stacionární AR(p) model můžeme přepsat na MA(∞) model a pokud MA(q) proces je invertibilní, poté ho můžeme přepsat na AR(∞). [25]

2.3.4.3 ARMA, ARIMA modely

Pokud bychom chtěli kombinovat oba předchozí modely, můžeme použít **autoregresní model klouzavých průměrů**, který je definován takto:

▶ **Definice 2.19.** ***Autoregresní model klouzavých průměrů*** *řádů p a q, zkráceně ARMA(p,q), je definován jako*

$$Y_t = c + \Phi_1 Y_{t-1} + \Phi_2 Y_{t-2} + \ldots + \Phi_p Y_{t-p} + \Theta_1\epsilon_{t-1} + \Theta_2\epsilon_{t-2} + \ldots + \Theta_q\epsilon_{t-q} + \epsilon_t = \\ = c + \sum_{i=1}^{p} \Phi_i Y_{t-1} + \sum_{i=1}^{q} \Theta_i\epsilon_{t-1}$$

, kde $[c, \Phi_1, \ldots, \Phi_p, \Theta_1, \ldots, \Theta_q]$ *jsou neznámé parametry a* ϵ_t *je bílý šum*

Tyto modely jsou populární například v ekonometrii při popisu ceny akcií, kde AR část popisuje vývoj na základě předchozích cen a MA část modeluje náhodné šoky. [27]

Pokud bychom chtěli model ještě víc rozšířit, můžeme k tomu využít ARIMA model, který je jednoduše ARMA model na d-krát diferencovaných datech.

▶ **Definice 2.20.** ***Autoregresní integrovaný model klouzavých průměrů*** *řádů p,d a q, zkráceně ARIMA(p,d,q), je definován jako*

$$Y_t^{'} = c + \Phi_1 Y_{t-1}^{'} + \Phi_2 Y_{t-2}^{'} + \ldots + \Phi_p Y_{t-p}^{'} + \Theta_1\epsilon_{t-1} + \Theta_2\epsilon_{t-2} + \ldots + \Theta_q\epsilon_{t-q} + \epsilon_t = \\ = c + \sum_{i=1}^{p} \Phi_i Y_{t-1}^{'} + \sum_{i=1}^{q} \Theta_i\epsilon_{t-1},$$

kde $[, \Phi_1, \ldots, \Phi_p, \Theta_1, \ldots, \Theta_q]$ *jsou neznámé parametry,* ϵ_t *je bílý šum a* $Y_t^{'}$ *je d-krát differencovaná řada*

Speciálně pak existují případy ARIMA modelů, které mají známé vlastnosti nebo charakteristiky:

- ARIMA(0,0,0) bez konstanty – bílý šum
- ARIMA(0,1,0) bez konstanty – náhodná procházka
- ARIMA(0,1,0) bez konstanty – náhodná procházka s driftem
- ARIMA(p,0,0) – autoregresní model řádu p
- ARIMA(0,0,q) – model klouzavých průměrů řádu q

Kapitola 3

Existující řešení

V této kapitole se budeme zabývat existujícími řešeními pro analýzu sentimentu. Ukážeme si různé nástroje a balíčky v různých programovacích jazycích. V poslední řadě se pak podíváme na existující řešení v komunikačních aplikacích.

Hojně využívaný jazyk pro strojové učení a datové vědy je programovací jazyk Python. Pro analýzu sentimentu zde můžeme tedy najít pár různých řešení, například NLTK nebo TextBlob, které se nezabývají jen analýzou sentimentu, ale obecně zpracováním přirozeného jazyka. I v dalších programovacích jazycích najdeme balíčky pro analýzu sentimentu.

3.1 NLTK

NLTK (Natural Language Toolkit) je nástroj pro zpracování přirozeného jazyka pomocí jazyka Python. Nabízí jednoduché použití různých rozhraní pro tokenizaci, stemming, parsování a sémantické uvažování [30]. Jedním z modulů s názvem *sentiment_analyzer* se používá, jak název napovídá, pro analýzu sentimentu. Obsahuje již předtrénovaný klasifikátor Vader, který funguje na základě slovníku a lexikálních pravidel. Při použití natrénovaného modelu jako výstup vrátí čtyři čísla (viz. kód 3.1). První tři složky označují negativní, neutrální a pozitivní skóre textu a nasčítávájí se do 1. Poslední složka, *compound*, není jen průměrem těchto předchozích tří složek, ale suma ohodnocení jednotlivých slov na základě pravidel modelu normalizovaná do intervalu $(-1, 1)$ [30].

Výpis kódu 3.1 Použití a výstup sentimentové analýzy pomocí NLTK

```
from nltk.sentiment import SentimentIntensityAnalyzer

SentimentIntensityAnalyzer().polarity_scores(text)

>{'neg': 0.123, 'neu': 0.841, 'pos': 0.015, 'compound': 0.0356}
```

Velmi pozitivní	4
Pozitivní	3
Neutrální	2
Negativní	1
Velmi negativní	0

Tabulka 3.1 Přiřazení číselných hodnot analýzy k třídám.

3.2 TextBlob

Dalším balíčkem v jazyce Python je TextBlob. Základy tohoto balíčku stojí na NLTK a nabízí mnoho metod pro zpracování přirozeného jazyka: stemming, parsování, integrace modelu WordNet, korekce hláskování a samozřejmě analýzu sentimentu [31]. Narozdíl od NLTK vrací pouze dvě složky, pozitivní a negativní (pravděpodobnosti klasifikace do určité třídy), a celkovou klasifikaci textu (viz. kód 3.2). V kódu můžeme při inicializaci modelu vidět parametr *analyzer*, kde si můžeme zvolit mezi dvěma konkrétními modely. Buď to *PatternAnalyzer*, založený na knihovně *pattern* nebo *NaiveBayesAnalyzer* z NLTK, natrénovaný na datasetu filmových recenzí [31].

Výpis kódu 3.2 Použití a výstup sentimentové analýzy pomocí TextBlob

```
from textblob import TextBlob
from textblob.sentiments import NaiveBayesAnalyzer

blob = TextBlob(text, analyzer).sentiment

> Sentiment(classification='pos',
            p_pos=0.7996209910191279,
            p_neg=0.2003790089808724)
```

3.3 Ostatní programovací jazyky

Python není jediným jazykem, který má nástroje na analýzu sentimentu. Jazyk C# od společnosti Microsoft obsahuje knihovnu ML.NET[1]. Funkce na analýzu vrací výstup ve stejném formátu jako balíček TextBlob.

Jazyk Java také disponuje balíčkem pro analýzu sentimentu, která využívá modul Stanford CoreNLP [2]. Výstupem zde je však celé číslo od 4 do 0, a v tabulce 3.1 můžeme vidět jejich význam.

3.4 Ostatní nástroje

Mimo knihovny a balíčky v programovacích jazycích existují i hotové nástroje, které jsou připraveny ihned k použití. Většina podporuje import dat z různých aplikací nebo sociálních sítí a zobrazení

[1] Více detailů zde: https://learn.microsoft.com/en-us/dotnet/machine-learning/tutorials/sentiment-analysis
[2] Dostupné z: https://blogs.oracle.com/javamagazine/post/java-sentiment-analysis-stanford-corenlp

různých grafů nebo přehledů. Tyto nástroje se poté většinou používají pro zjištění nálady na sociálních sítích či vlastních webech nebo blozích [32]. Jednotlivé nástroje nebudeme nijak popisovat, protože to není v rozsahu této práce, pouze ukážeme pár možností s odkazy na jednotlivá řešení [32]:

- MonkeyLearn – https://monkeylearn.com/
- Lexalytics – https://www.lexalytics.com/semantria/
- Brandwatch – https://www.brandwatch.com/
- Rosette – https://www.rosette.com/capability/sentiment-analyzer/
- A spousta dalších.

3.5 Aplikace v komunikačních aplikacích

Mimo zmíněných nástrojů v kapitole 3.4 výše, můžeme v komunikačních aplikacích využívat i aplikace specializované, oficiálně vydané, na konkrétní platformu.

Aplikace Slack nabízí možnost integrace aplikací od jiných vývojářů, které můžeme najít na Slack Apps Directory. Za zmínku stojí například **Anonymous Feedback Bot by OpenSay**[3], která mimo analýzy sentimentu nabízí mnoho dalších funkcí a **ChatAcuity**[4].

Na další populární komunikační platformě Discord pak existuje open-source řešení Monitori [5], které nabízí podobné funkcionality jako výše uvedené aplikace.

[3] Anonymous Feedback Bot by OpenSay, https://slack.com/apps/AH64TE3NC-anonymous-feedback-bot-by-opensay?tab=more_info

[4] ChatAcuity, https://slack.com/apps/AKWJUT1T6-chatacuity?tab=more_info

[5] https://github.com/ajyuan/Monitori

Kapitola 4

Implementace a výsledky

V této kapitole se budeme zabývat implementací nástoje na sentimentovou analýzu, hledáním vhodného modelu pro její klasifikaci a modelováním časové řady z dat získaných z analýzy. Podrobně rozebereme výsledky jednotlivých modelů a výsledný model porovnáme s existujícím řešením.

Cílem této práce je analyzovat náladu v komunikačních aplikacích, při použití vlastních metrik a poté predikovat její vývoj. Celá tato analýza je pak implementována v rámci bota v aplikaci Slack (více o Slacku viz sekce 4.1). K analýze nálady budeme potřebovat najít vhodný model. Analýza sentimentou je uváděna jako klasifikační úloha, kde stačí rozlišit pozitivní a negativní texty [1], ale pro náš případ zde budeme uvažovat regresní úlohu, abychom mohli predikovat vývoj nálady.

4.1 Slack

Slack je komunikační platforma uzpůsobená pro komunikaci v rámci společnosti, firmy nebo pracovní skupiny. Je využívána například společnostma OpenAI, Revolut, Uber, IBM a mnoho dalšími [33]. Slack nabízí vytvoření vlastního prostoru (workspace), kde se mohou vytvářet různé místnosti/kanály ke komunikaci. Nepoužívá se zde jen text, ale i obrázky, videa nebo jiné soubory. Zprávy se dají uspořádávat do vláken a každý může na zprávu reagovat vestavěnou nebo vlastní reakcí, což jsou většinou různé emotikony. V neposlední řadě Slack umožňuje integraci aplikací, a to jak známých jako jsou Google Drive nebo Office 365, tak i vlastnoručně napsané, které využívají Slack API.

4.1.1 Slack API

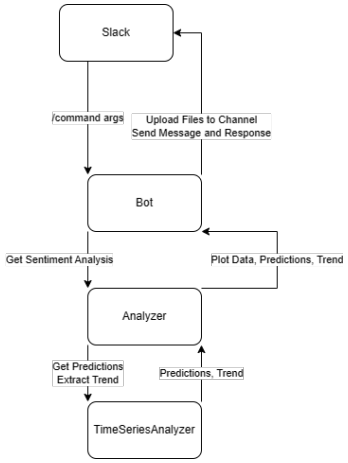
Ke komunikaci s vlastní aplikací můžeme využít několik způsobů, a nejčastěji to jsou příkazy (slash commands) nebo eventy. Pro naší práci se nejvíce hodí příkazy, což vlastně jsou zkratky pro konkrétní akce ve Slacku. Můžeme je použít tak, že do pole pro zprávu napíšeme znak / následovaný specifickými klíčovými slovy. Jako příklad můžeme uvést */mute*, který ztlumí

aktuální kanál, a */remind*, který nastaví připomenutí pro sebe nebo ostatní. Příkazy po zadání a odeslání fungují tak, že obsah je odeslán prostřednictvím HTTP POST požadavku do aplikace a poté aplikace nějakým způsobem odpoví [34]. Po definici příkazu může uživatel zadat libovolný text, který pak může sloužit jako argumenty k příkazům.

Aplikace pak musí být ověřená specifickým tokenem a pro používání metod Slack API musí mít definované dostatečná práva (scopes). Podrobnější informace lze nalézt v oficiální dokumentaci viz [34].

4.2 Specifikace problému

Jak již bylo řečeno v úvodu této kapitoly, výstupem této práce je aplikace integrovaná do platformy Slack (reagující na příkazy), která analyzuje náladu. Na obrázku 4.1 můžeme vidět schéma znázorňující flow celého programu. Bot se stará o komunikaci se Slack API a data z této API (historie zpráv v kanále) posílá dál k analýze. Analyzer pak provede analýzu sentimentu (viz kapitola 4.6.2) a analýzu nálady v čase (viz kapitola 4.5) pomocí TimeSeriesAnalyzer. Bot poté získá vráceno 3 grafy a jednoduché statistiky, které nahraje na Slack do požadovaného kanálu.



Obrázek 4.1 Diagram ukazující flow programu.

Analyzer používá model na zjištění nálady. Konkrétní informace o modelu a o hledání nejlepšího si řekneme v následujících kapitolách.

4.2.1 Klasifikace vs. regrese

Problém analýzy sentimentu je klasifikační problém, kde klasifikujeme text jako negativní nebo pozitivní [1]. Pro náš případ je však vhodnější se na problém koukat jako na regresní problém, kvůli predikci a analýze. Zavedeme tedy místo klasifikačních tříd, 0 = pozitivní a 1 = negativní, novou spojitou veličinu (označme ji S), která bude v intervalu od -1 do 1, kde 1 je nejpozitivnější, 0 neutrální a -1 nejnegativnější sentiment. Tuto novou veličinu budeme určovat pomocí regresních modelů nebo pomocí pravděpodobnostních klasifikačních modelů, ze kterých získáme

pravděbodobnost $P(Y = 0)$ a tu jen transformujeme na naši novou veličinu S:

$$S = 2 \cdot P(Y = 0) - 1.$$

4.3 Model pro zjištění nálady

Modely použijeme z Pythonovského balíčku *scikit learn*. Z téhož balíčku pak použijeme třídu *ParameterGrid*, která nám pomůže s laděním hyperparametrů. Jednoduše můžeme tak specifikovat různé hodnoty hyperparametrů a *ParameterGrid* se postará o jejich kombinace.

V následujících podkapitolách si představíme použité modely a jejich hyperparametry, které budeme ladit. Teoretický základ k následujícím modelům můžeme najít v kapitole 2.2.5. Všechny parametry jsou z dokumentace *sklearn* [14]

4.3.1 KNeighborsRegressor

- **n_neighbors** – integer – Počet nejbližších sousedů, které budou použity pro regresi.
- **weights** – 'uniform', 'distance' – Váhy použité při regresi, kde 'uniform' uvažuje stejné váhy, ale 'distance' přiřazuje váhu na základě vzdálenosti ($w = \frac{1}{d}$, kde d je vzdálenost souseda) tak, že bližší sousedi mají větší vliv na regresi. Jde definovat i vlastní funkce.
- **algorithm** – Algoritmus použitý k vypočtení nejbližších sousedů.
- **metric** – Metrika použitá k vypočtení vzdálenosti mezi dvěma body.
- **p** – integer – Parametr p pro Minkovského metriku.

4.3.2 LinearSVR

- **epsilon** – float – Parametr epsilon ve ztrátové funkci.
- **loss** – 'epsilon_insensitive', 'squared_epsilon_insensitive' – Ztrátová funkce.
- **C** – float – Parametr regularizace. Síla regularizace je nepřímo úměrná parametru C.
- **fit_intercept** – bool – Značí zda se má pro tento model vypočítat intercept.

4.3.3 DecisionTreeRegressor

- **criterion** – 'squared_error', 'friedman_mse', ... – Kritérium měřící kvalitu rozdělění dat.
- **max_depth** – integer – Maximální hloubka rozhodovacího stromu.
- **splitter** – 'best', 'random' – Určuje způsob rozdělení dat v každém vrcholu, kde jména jsou poměrně samovysvětlující.
- **min_samples_split** – Minimální počet dat, při kterém se bude vrchol dělit.
- A další.

4.3.4 LogisticRegression

- **solver** – ‘lbfgs’, ‘newton-cholesky’, ... – Metoda použitá k řešení optimalizačního problému.
- **fit_intercept** – bool – Značí jestli bude intercept přidán do rozhodovací funkce.
- **C** – float – Inverzní hodnota síly regularizace.

4.3.5 BernoulliNB

- **alpha** – float/pole – Vyhlazovací parametr.
- **fit_prior** – bool – Značí jestli se model má naučit apriorní
- **class_prior** – pole – Apriorní pravděpodobnosti tříd

4.3.6 ComplementNB

- **alpha** – float/pole – Vyhlazovací parametr.
- **fit_prior** – bool – Značí jestli se model má naučit apriorní
- **class_prior** – pole – Apriorní pravděpodobnosti tříd

4.3.7 GaussianNB

- **priors** – pole – Apriorní pravděpodobnosti tříd
- **var_smoothing** – float – Část největší odchylky všech prvků, která se přičítá k odchylkám kvůli stabilitě výpočtu.

4.3.8 MLPClassifier

- **hidden_layer_sizes** – pole integerů – Délka pole specifikuje počet skrytých vrstev a jednotlivé prvky pak počet neuronů v konkrétní skryté vrstvě.
- **activation** – ‘identity’, ‘logistic’, ‘tanh’, ‘relu’ – Aktivační funkce pro skrytou vrstvu.
- **solver** – Metoda použitá k řešení optimalizaci vah.
- **learning_rate** a **learning_rate_init** – Počáteční hodnota a následný „průběh“ koeficientu učení.
- **max_iter** – integer – Maximální počet iterací.

4.3.9 Data

K natrénování specifického modelu, kdy uvažujeme supervizované učení, budeme potřebovat data, kde známe výstupní veličinu. Datasetů k sentimentové analýze existuje relativně dost (okolo 1200 na Kaggle [35]) a můžeme je rozdělit na několik kategorií:

- Recenze produktů – Obsahuje recenze nejrůznějších produktů ze známých serverů (např. Amazon, IMBD a další). Některé jsou klasifikovány pomocí dalších dat z recenzí (např. hvězdičky, skóre) a některé jsou klasifikované na základě reálného sentimentu.
- Sociální sítě – Příspěvky ze sociálních sítí, nejčastěji se zde objevuje Twitter.
- Specifická témata – Často (ne vždy) se jedná o příspěvky ze sociálních sítí, které se týkají jednoho specifického tématu (např. finance, kryptoměny, COVID a další).

Datasety můžeme najít ve specifických jazycích, ale nám postačí angličtina. Jelikož budeme zjišťovat náladu v komunikačních aplikacích, nejvíce podobná budou data ze sociálních sítí, jelikož nebudeme vytvářet nástroj pro analýzu v konkrétní oblasti. Kdyby ano, mohli bychom přemýšlet o použití datasetu, který je z této konkrétní oblasti. Zvolíme tedy data ze sociální sítě Twitter a konkrétní dataset je pak dostupný zde na Kaggle.

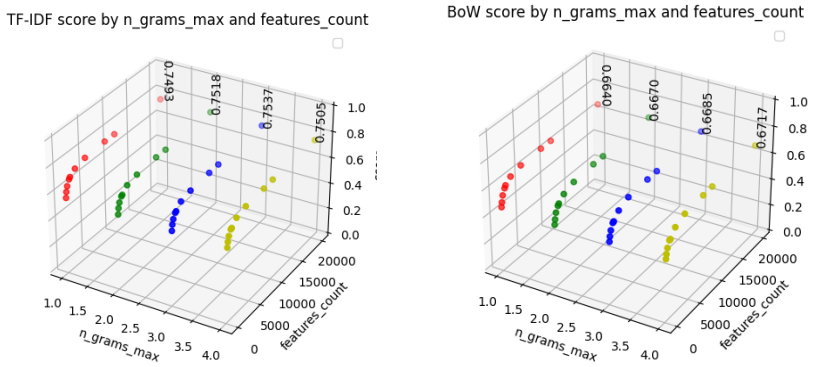
Text	Category
family mormon have never tried explain them th...	0
buddhism has very much lot compatible with chri...	0
seriously don say thing first all they won get ...	1
for your own benefit you may want read living b...	0
you should all sit down together and watch the ...	1

Tabulka 4.1 Ukázka datasetu.

4.3.9.1 Zpracování textu

K trénování modelu musíme jednotlivé texty očistit a zpracovat. Zprávy na Twitteru (a i poté na Slacku) obsahují označení uživatelů: znak „@“ a hned za ním uživatelské jméno. Tato informace nám nijak nepomůže k analýze sentimentu. Stejně nám nepomůžou všechny ne-alfanumerické znaky, které také odstraníme. Dále pak odstraníme slova kratší jak 3 znaky, která většinou jsou zájmena, předložky nebo spojky. Posledním krokem je stemming, který nám pomůže rozlišovat slova na základě významu a ne tvaru. Pokročilejší technika, lemmatizace, se v našem případě nevyplatí, kvůli výkonu, protože budeme následně dělat analýzu na vyžádání, kde musíme dělat ústupky ve prospěch výkonu. Takto zpracovaný text musíme přetransformovat na příznaky a k tomu nám pomůže Bag of Words nebo TF-IDF. Z Pythonovského balíčku *scikit learn* tomu odpovídají třídy *CountVectorizer* a *TfidfVectorizer*.

Opět zde můžeme najít hyperparametry, které můžeme nějakým způsobem ladit. Pro oba třídy jsou stejné:



(a) TF-IDF **(b)** BoW

Obrázek 4.2 Porovnání vlivu parametrů **max_features** a **ngram_range** na výkonnost modelu.

- **stop_words** – 'english' – Slova, která nemají význam pro pochopení smyslu věty a budou odstraněna.
- **ngram_range** – (min_n, max_n) – Rozsah pro délku n-gramů, které budou extrahovány.
- **max_features** – integer – Maximální počet příznaků, které budou extrahovány.
- A další.

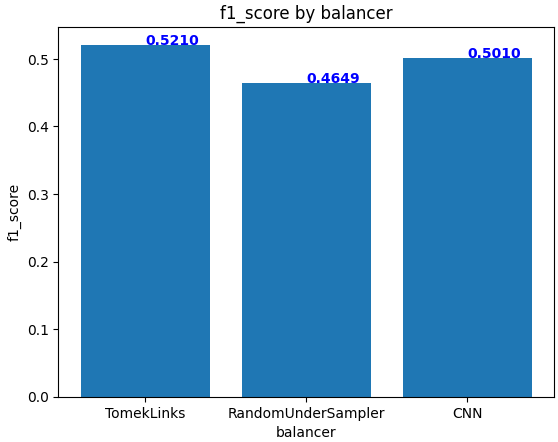
Zkusíme tedy různé kombinace parametrů, konkrétně **ngram_range** a **max_features**. Na grafech 4.2 můžeme vidět, že vyšší počet příznaků má pozitivní vliv na výkon a nejlépe na tom je při maximální délce ngramu rovno 3 (minimum bylo vždy 1, protože výsledky při použití pouze bigramů nebo trigramů byly velmi nízké a v grafu nejsou, protože by tento parametr přidal další dimenzi grafu). Celkově je na tom lépe *TfidfVectorizer*, proto budeme do budoucna pracovat s ním.

4.3.9.2 Balancování

Jelikož třídy v datech nejsou zastoupeny rovnoměrně, budeme uvažovat o balancování dat. Dat máme relativně dost, můžeme tedy použít undersamplery. Nyní využijeme Pythonovský balíček *imblearn*, který poskytuje různé algortimy pro balancování dat.

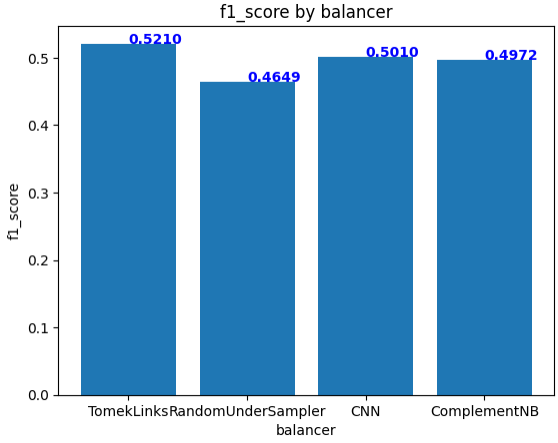
Budeme tedy porovnávat **RandomUnderSampler**, **TomekLinks** a **CondensedNearestNeighbour**. Zde nebudeme ladit hyperparametry (teoreticky bychom mohli u CNN), ale jen porovnávat různé metody. Všechny mají společný parametr **sampling_strategy**, který určuje jaké třídy mají být *přesamplovány*. Jelikož se v našem případě jedná o problém binární klasifikace a používáme undersampling, jako hodnotu tohoto parametru použijeme *majority* (nebo taky ekvivalentní pro náš případ *not minority*), který zajistí odstranění dat pouze z majoritní třídy.

Na grafu 4.3 můžeme vidět porovnání jednotlivých metod, kde nejlépe dopadl balancer TomekLinks.



Obrázek 4.3 Porovnání metod pro balancování.

V teoretické části jsme ukázali, že některé modely jsou designové na pracování s nevybalancovanými datasety. Zkusíme tedy porovnat, jak se tento model s daty dokáže vypořádat. Konkrétně jde o Complement Naive Bayes. Na grafu 4.4 vidíme, že sice výsledky měl podobné, ale TomekLinks pořád vychází nejlépe.



Obrázek 4.4 Porovnání metod pro balancování s ComplementNB.

4.3.10 Experiment a výsledky

Po zpracování textu, očištění a balancování dat (viz kapitola 4.3.9) můžeme přistoupit k otestování různých modelů s různými parametry. Konkrétně budeme testovat všechny modely z kapitoly 4.6.2. Jejich přesnost pak budeme ověřovat pomocí křížové validace. Díky ní získáme několik metrik, které budeme porovnávat a na jejich základě pak vybereme vhodný model pro náš případ.

Výpis kódu 4.1 Python kód pro trénování a evaluaci všech modelů

```
# imports ...
```

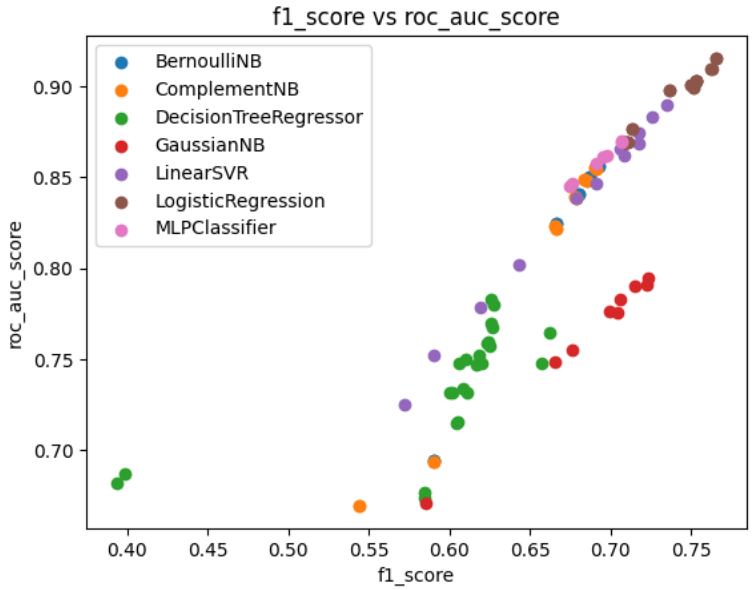
```
models = [
    ( model_class, {
            'param_name' : [param_val1, ... , param_valn],
            ... }),]

# Load data and transform using TD-IDF
df = load_and_clean_data(data.csv)

X_train, X_test, y_train, y_test = train_test_split(
df, df['output'],test_size=0.2)

X_train_balanced, y_train_balanced = balance_data(
X_train, y_train,
balancer= TomekLinks(sampling_strategy='not␣minority'))

for model_class, params in models:
    parameter_grid = ParameterGrid(params)
    for p in parameter_grid:
        model = model_class(p**)
        score = cross_validate(
        model, X_train_balanced, y_train_balanced,
        scoring=['f1_score', 'roc_auc_score'])
```

V kódu **??** můžeme vidět celý průběh evaluace všech modelů, kde jen v poli *models* jsou konkrétní třídy modelů a konkrétní parametry, které chceme zkoušet. V proměnné *score* pak budou pole námi určených metrik, kde každá hodnota v poli odpovídá jednomu běhu křížové validace. Pro srovnání modelů budeme pracovat s průměrem těchto hodnot.

Na grafu 4.5 můžeme vidět jednotlivé modely a jejich F1 a ROC-AUC skóre. Každá barva zde reprezentuje druh modelu a jednotlivé body jsou pak různá nastavení hyperparametrů. Některé hyperparametery neměly na skóre vliv, což z grafu nejde moc dobře vyčíst. Také můžeme pozorovat, že jednotlivé modely MLPClassifier, LogisticRegression, GaussianNB a DecisionTreeRegressor jsou v grafu u sebe (až na vyjímky), tudíž změna hyperparametrů neměla na ně nijak velký vliv. Na druhou stranu ComplementNB, LinearSVR a BernoulliNB vykazují velkou závislost skóre na jejich hyperparametrech. Na první pohled vypadá, že nejlépe se dařilo logistické regresi, ale zkusíme se na graf podívat detailněji.

Na grafu 4.6 jsme omezili obě skóre zdola na pro nás zajímavější hodnoty. Jak lze vidět, skutečné logistická regrese dopadla nejlépe (v grafu ynak trojúhelníku směřující doleva) pro kombinaci hyperparametů s parametrem $C = 0.5$. Dále v pořadí následuje LinearSVR, který pro nějaké hyperparametry byl srovnatelný s MLP klasifikátorem a logistickou regresí s parametrem $C = 5$. Z těchto modelů, nehledě na hyperparametry, pak nejhůře dopadl GaussianNB, který ale pořád dosáhl relativně slušných výsledků.

Průměry skóre přes všechny kombinace hyperparametrů můžeme vidět na grafu 4.7 a 4.8, kde opět nejlépe dopadla logistická regrese. V průměru nejhůře pak dopadl rozhodovací strom.



Obrázek 4.5 F1 skóre a ROC-AUC všech modelů

Na grafech **??** a 4.10 můžeme vidět i jiné metriky a to konkrétně na hodnoty z matice záměn.

4.3.10.1 Výsledný model

Ve všech metrikách se zdál nejlepší model logistické regrese. Parametry modelu výsledné metriky neovlivňovaly tolik, ale přeci k nějakému zlepšení došlo. Pokud se ale podíváme na SVR, vidíme že parametry modelu výsledné metriky ovlivnily mnohem více. Zkusíme tedy znova ladit hyperparametry, ale jen těchto dvou zmíněných modelů.

Spustíme tedy experiment znovu, jen s více parametry. Na grafu 4.11 můžeme opět vidět srovnání těchto modelů na základě F1-skóre a ROC-AUC.

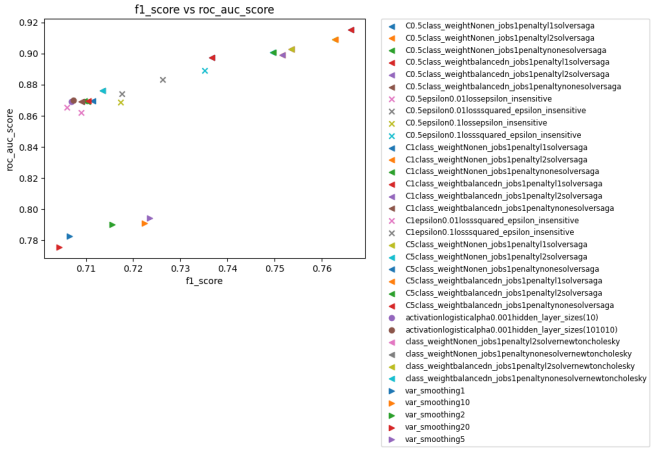
Opět vidíme, že většina modelů logistické regrese je na tom lépe než SVR, a to v obou metrikách. Při přiblíženém pohledu na grafu 4.12, ale zjistíme, že se mezi nimi nachází i pár modelů SVR.

Metrika ROC-AUC značí schopnost separovat třídy a z modelů dostáváme pravděbodobnost p jestli datový bod patří do třídy 0, ze které dostaneme výslednou třídu následovně:

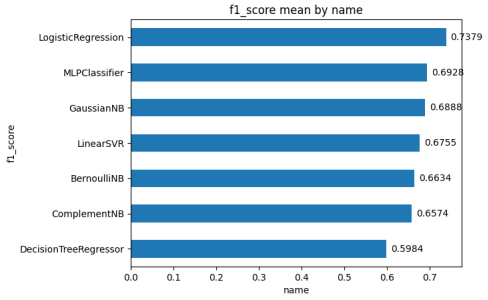
$$\hat{Y} = \begin{cases} 0, & P \leq p \\ 1, & \text{jinak} \end{cases},$$

kde P je získaná pravděpodobnost z modelu a p je prahová hodnota, kterou jsme doteď uvažovali 0.5. Nyní zkusíme hýbat touto prahovou hodnotou a podíváme se na její vliv na skóre. Tuto metodu zkusíme pouze na dvou nejlepších modelech. Výsledky můžeme vidět na grafu **??**. Vidíme zde, že lépe je na tom opět logistická regrese a pro oba modely zde byla lepší prahová hodnota větší než 0.5, a to konkrétně 0.53 resp. 0.51 pro logistickou regresi resp. SVR.

Ze všech experimentů nám vyšel jako nejlepší model logistické regrese. Jeho inicializaci a



Obrázek 4.6 F1 skóre a ROC-AUC modelů se zoomem.



Obrázek 4.7 Průměr F1 skóre různých druhů modelů.

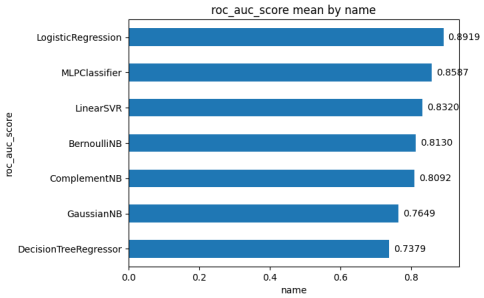
konkrétní parametry můžeme vidět v kódu 4.2.

Výpis kódu 4.2 Inicializace a parametry modelu MODEL

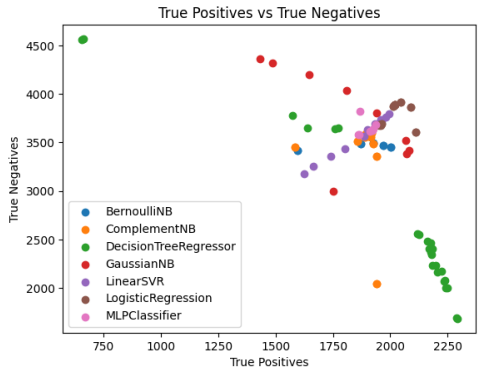
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(penalty='l1',
                           C=1,
                           class_weight=None,
                           n_jobs = -1,
                           solver = 'saga')
```

Finální metriky tohoto modelu zjistíme pomocí testovacích dat. Výsledky jsou zanesené v tabulce 4.2

V kapitole 3 jsme si také ukázali existující řešení v jazyce Python, konkrétně NLTK a TextBlob. Zkusíme výsledky našeho modelu postavit proti těmto modelům. Na grafu 4.14 můžeme vidět absolutní chybu predikcí. Náš model má relativně srovnatelné výsledky s klasifikátorem z balíčku TextBlob a hůře na tom je v porovnání s NLTK. Jak si ale můžeme všimnout, model TextBlob v porovnání s NLTK má podobnou chybu jako náš model.



Obrázek 4.8 Průměr ROC-AUC různých druhů modelů.



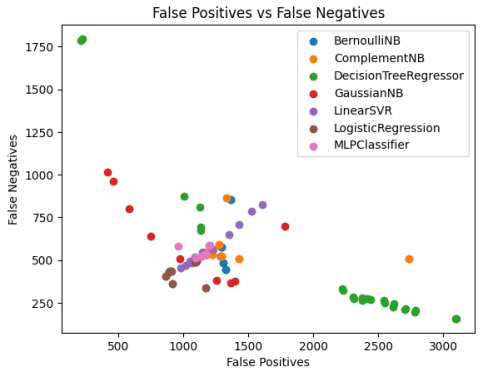
Obrázek 4.9 Graf skutečně pozitivních a skutečně negativních predikcí.

Metrika	Hodnota
F1 Score	0.785412
ROC-AUC	0.914432
Accuracy	0.837942
Average Precision	0.848132
True Positive	956.000000
False Positive	305.000000
False Negative	281.000000
True Negative	2074.000000

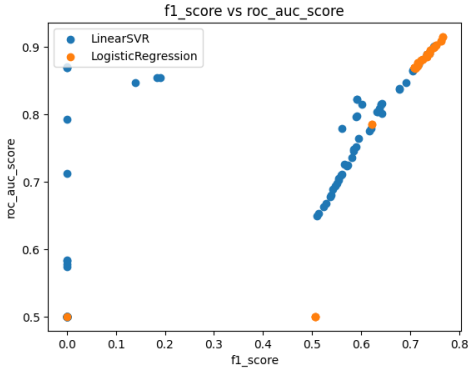
Tabulka 4.2 Metriky finálního modelu na testovacích datech.

4.4 Vlastní metriky

Již v kapitole 4.2.1 jsme si ukázali, jak se postavíme k ohodnocení textu na základě sentimentu. Pro připomenutí jsme si zavedli novou veličinu S, která bude v intervalu $\langle -1, 1 \rangle$, kde 1 je nejpozitivnější, 0 neutrální a -1 nejnegativnější sentiment. Jelikož budeme analyzovat náladu v komunikačních aplikacích, můžeme využít i dalších nástrojů, než jen textu. Na Slacku můžeme tedy využít např. reakce na zprávy, které také nějak reflektují zabarvenost zprávy. Jméno reakce (které dostaneme ze SlackAPI) nemusí nutně vhodně reprezentovat sentiment při jejím použití, ale ve velké části případů to tak je. Kvůli tomu implementujeme možnost vlastní konfigurace, jak



Obrázek 4.10 Graf falešně pozitivních a falešně negativních predikcí.



Obrázek 4.11 F1-skóre a ROC-AUC logistické regrese a SVR podruhé.

má program pracovat (jedná se o soubor s páry klíč-hodnota, kde klíč je název reakce a hodnota je desetinné číslo od -1 do 1).

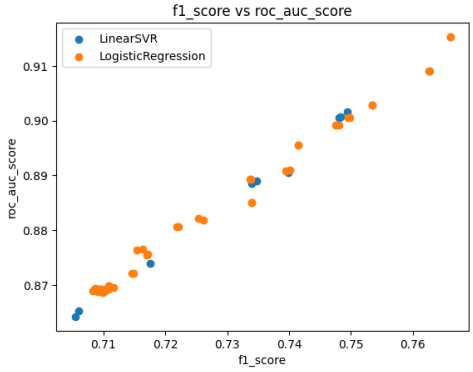
K použití reakcí v naší analýze sentimentu jsme našli několik přístupů.

Předpokládejme tedy, že analyzujeme zprávu, na kterou nějací uživatelé reagovali pomocí reakcí. Pak tedy můžeme využít:

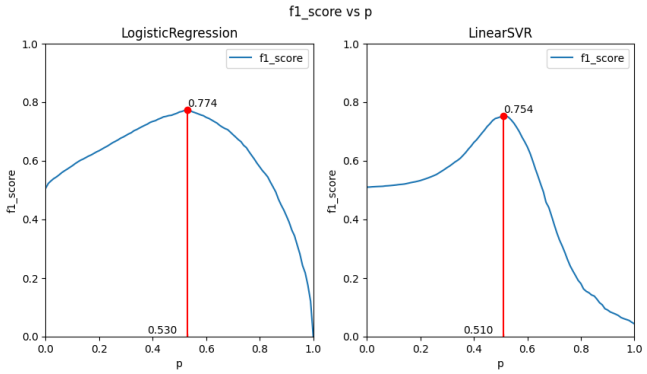
- **Přístup 1** – Za text zprávy připojíme jména jednotlivých reakcí, a celý tento text analyzujeme. Pokud se jednalo o pozitivní reakce, měli bychom dostat pozitivnější výsledek. Tento přístup ale přináší několik problémů. Prvním je, že vůbec nereflektuje počet stejných reakcí na zprávě. Reakce pak mají stejnou váhu, nehledě na jejich počet. Dalším problémem je možné úplné vytracení vlivu původní zprávy, což určitě není žádané chování. To může nastat, pokud je zpráva moc krátká nebo reakcí je příliš mnoho (nebo obojí dohromady).

- **Přístup 2** – Modifikujeme první přístup, kdy stejně budeme připojovat jména reakcí za text, který potom bude analyzován, ale jména připojíme tolikrát, kolik bylo reakcí na zprávě. Tím eliminujeme první problém předchozího přístupu, ale druhý problém zůstává, ba je i mnohem horší.

- **Přístup 3** – Tentokrát budeme text a reakce analyzovat zvlášť. Z analýzy dostaneme dvě hodnoty sentimentu tj. číslo od -1 do 1. Čísla pak můžeme sečíst a opět *vrátit* do intervalu



Obrázek 4.12 F1-skóre a ROC-AUC logistické regrese a SVR podruhé.



Obrázek 4.13 Vliv prahové hodnoty p na F1-skóre.

$\langle -1, 1\rangle$ (vydělením 2).

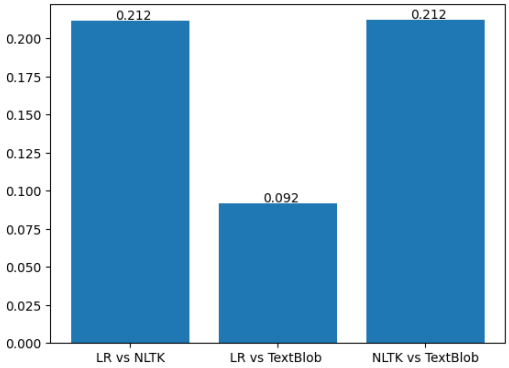
- **Přístup 4** – Jedná se o vylepšení 3. přístupu, kde měly reakce a zpráva stejnou váhu. Tentokrát si zvolíme váhu pro reakce, kterou označíme $w \in \langle 0, 1\rangle$, čímž získáme i váhu pro zprávu $1 - w$. Výsledek pak získáme převážením získaných výsledků analýz:

$$S = wS_{\text{reactions}} + (1 - w)S_{\text{message}}.$$

- **Přístup 5** – Jedná se o vylepšení 4. přístupu v rámci analýzy reakcí. V předchozích dvou přístupech jsme ignorovali celkový počet jednotlivých reakcí, jen nás zajímalo, jaké reakce jsou na zprávě. V tomto přístupu tento počet zahrneme do výpočtu, protože větší počet stejných reakcí značí nějakou shodu více uživatelů a je tudíž relevantnějším ukazatelem než samostatná reakce, která mohla být např. omyl nebo nepochopení. Vzorec pro celkový sentiment tedy zůstává stejný, jen změníme výpočet:

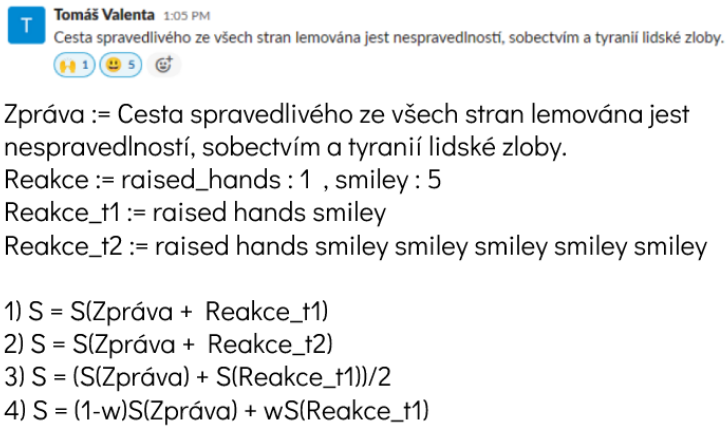
$$S_{\text{reactions}} = \sum_{i=0}^{N_r} \frac{n_i S_i}{N},$$

kde N_r je počet typů reakcí, n_i je počet i-té reakce na zprávě, N je celkový počet reakcí



Obrázek 4.14 MAE mezi predikcemi pravděpodobností jednotlivých modelů.

na zprávě a S_i je ohodnocení sentimentu i-té reakce, buď na základě analýzy nebo hodnoty z konfiguračního souboru.



Obrázek 4.15 Schéma rozdílu mezi prvními 4 přístupy. (5. je více rozepsaný v textu)

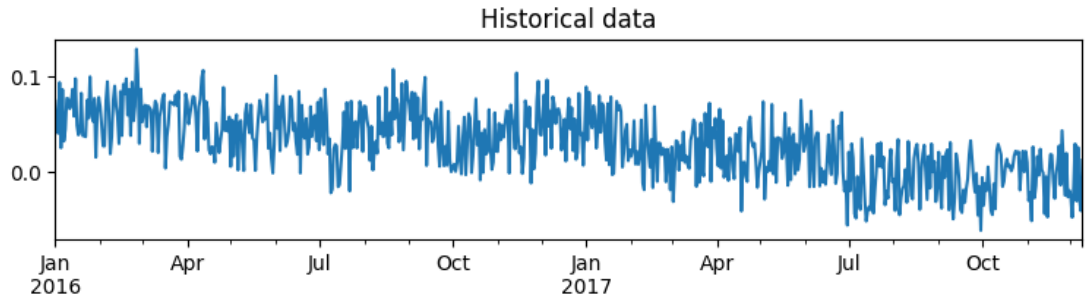
4.5 Analýza časové řady

Cílem této práce je náladu v komunikačních aplikacích nejen analyzovat, ale i získat trend nálady a nějakým způsobem predikovat náladu do budoucnosti. Jelikož se budou analyzovat zprávy, které jsou odeslány v určitý čas, můžeme k predikci a trendu využít aparátu časových řad. Jelikož modelování časových řad je většinou ruční práce, zkusíme nejdříve časovou řadu manuálně analyzovat a zjistit nějaké základní informace. Jelikož ale chceme dělat predikce na vyžádání, budeme muset tento proces automatizovat na úkor přesnosti.

K analýze budeme využívat Pythonovské balíčky *statsmodels*, který obsahuje všechny potřebné modely a funkce, a *pmdarima*, který nám pomůže při hledání vhodných parametrů modelů.

4.5.1 Manuální analýza

Jako data k analýze jsem použil zprávy ze Slacku Gitteru, online kurzu programování[1]. Data jsme zpracovali stejně jako v předchozí kapitole: odstranili jsme nealfanumerické znaky a označení, provedli stemming a transformovali text pomocí TF-IDF. Následně jsme každou transformovanou zprávu analyzovali modelem logistické regrese s parametry uvedených v kapitole 4.3.10.1. Jelikož v datasetu najdeme pouze zprávy, ale žádné reakce, budeme pracovat pouze s hodnotou sentimentu zprávy (tj. váha reakce je nula). Grafické znázornění těchto hodnot můžeme vidět na grafu 4.16.



Obrázek 4.16 Historické hodnoty sentimentu zpráv ze Slacku Gitteru.

Na první pohled vidíme

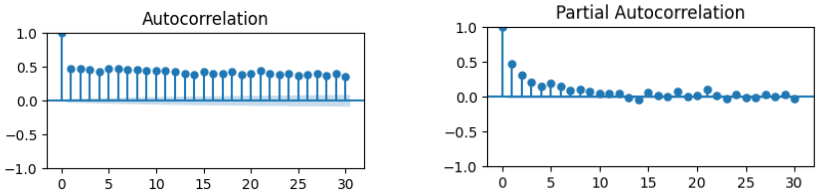
Při pohledu na ACF a PACF (grafy 4.17) můžeme na první pohled říci pár poznatků. PACF klesá k nule a funkce má výrazné peaky jen na začátku. To může značit neexistenci sezónnosti a vliv historie zpráv na náladu zhruba 4 dny nazpět. ACF pak začíná blízko k 1, ale pomalu klesá.

K získání více informací se můžeme podívat na dekompozici časové řady pomocí funkce *seasonal_decompose*, která nám časovou řadu dekomponuje na trend, sezónní složku a rezidua. Výsledek funkce je vidět na grafu 4.18, a můžeme z něho vyčíst zanedbatelnou sezónnost a přítomnost trendu.

K odstranění trendu můžeme využít první diference, tedy odečtení řady sama od sebe ve zpoždění jedna. Výsledná řada (viz graf 4.19) pak již neobsahuje trend a připomíná bílý šum.

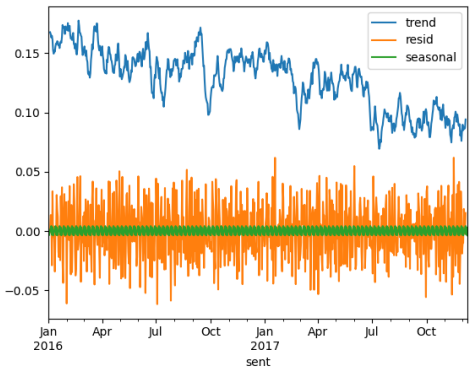
Opět se na grafech 4.22 můžeme podívat na ACF a PACF, kde ACF má výrazný peak v prvním zpoždění a PACF klesá k nule. Tyto znaky ukazují na MA(1) model, ale odhad pouze

[1]Dostupné z https://www.kaggle.com/datasets/freecodecamp/all-posts-public-main-chatroom

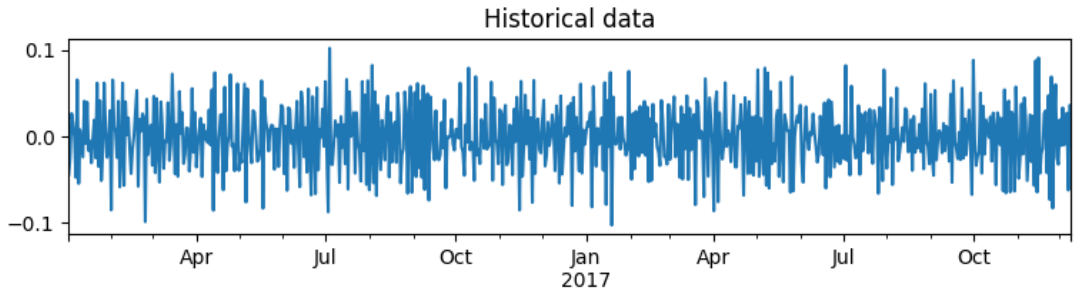


(a) ACF **(b)** PACF

Obrázek 4.17 ACF a PACF časové řady.



Obrázek 4.18 Dekompozice časové řady.

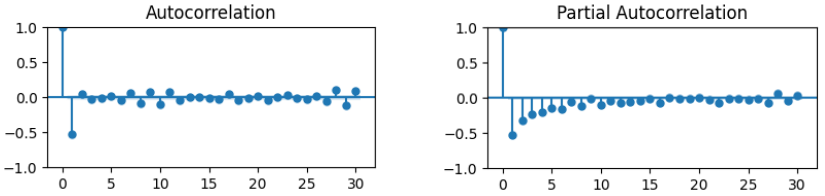


Obrázek 4.19 Diferencovaná časová řada hodnot sentimentu zpráv ze Slacku Gitteru.

vyčtením z grafů nemusí být vždy přesný, dokonce může být i zavádějící. Proto použijeme funkci *auto_arima* z již zmíněného balíčku *pmdarima*, která pomocí zvoleného informačního kritéria vyhledá nejlepší model pro naše data. V kódu 4.3 můžeme vidět její použití a parametry. Důležitými jsou intervaly řádů p a q, počet diferencí a sezónnost.

Výpis kódu 4.3 Ukázka použití, parametrů a výstupu funkce auto_arima

```
import pmdarima

best_model = pmdarima.auto_arima(data,
             start_p=1, start_q=1,
             max_p=5, max_q=5, max_d=1,
```



(a) ACF **(b)** PACF

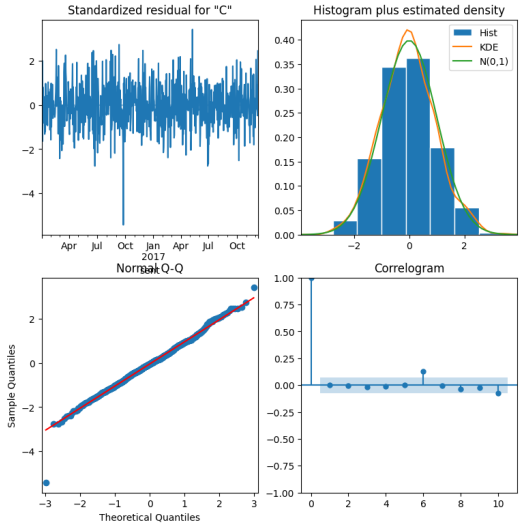
Obrázek 4.20 ACF a PACF diferencované časové řady.

```
            seasonal=False,
            information_criterion='aic',
            trace=True,stepwise=True)

>...
>Best model:  ARIMA(2,1,1)(0,0,0)[0]
>Total fit time: 5.273 seconds
```

Po nalezení řádů můžeme tedy vytvořit model a nafitovat jej. Implementace modelu umožňuje vykreslit diagnostické grafy, které nám umožňují zkontrolovat výsledný model, a konkrétně to jsou [36]:

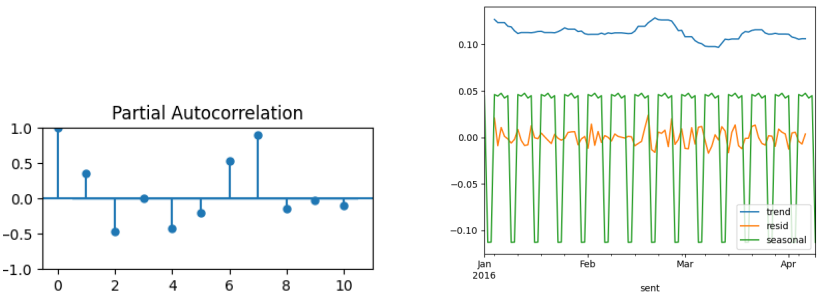
1. Standartizovaná rezidua v čase
2. Histogram a odhadnutá hustota standartizovaných reziduí
3. Q-Q graf
4. Korelogram

Diagnostiku pro náš konkrétní model můžeme vidět na grafu 4.21. V korelogramu nevidíme výraznější hodnoty a v grafu reziduí nejde vidět žádný vzor. To znamená, že model popisuje časovou řadu dobře a žádných jiných vzorů se už nemůžeme chytit.



Obrázek 4.21 Diagnostika ARIMA(2,1,1) modelu.

Model tedy máme natrénovaný a diagnostikovaný. Následně zkusíme dataset poupravit, aby více připomínal data z firemního prostředí, na kterých budeme celý náš program testovat. Ve firemním prostředí tedy můžeme očekávat až nulovou komunikaci o víkendu. Zkusíme tedy hodnoty sentimentu v sobotu a v neděli nastavit na 0 (viz kód 4.4), což může značit ryze neutrální náladu nebo chybějící data. Z logiky věci můžeme tušit v datech sezónnost s periodou sedmi dní. Zkusíme tedy naši teorii ověřit.



(a) PACF **(b)** Dekompozice

Obrázek 4.22 PACF a dekompozice upravené časové řady.

Výpis kódu 4.4 Simulace nulové komunikace o víkendu.

```
df['sentiment'] = df.apply(
lambda x: x['sentiment']  if x['sent'].weekday() < 5 else 0, axis=1)
```

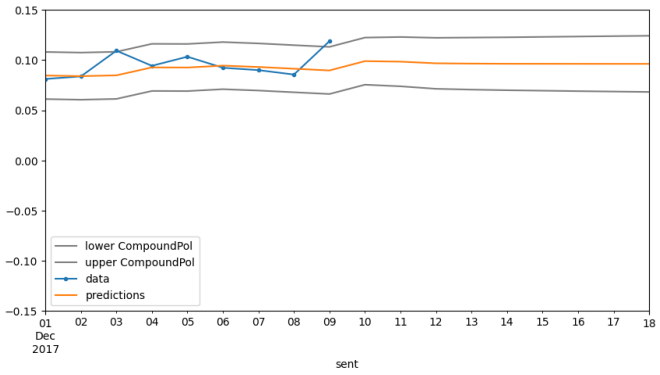
Na grafu 4.22 vidíme PACF, kde vidíme hodnotu blížící se k nule v sedmém zpoždění. To by odpovídalo naší teorii o sedmidenní periodě. Vedle grafu PACF můžeme vidět dekompozici této časové řady, kde jsme pro přehlednost zobrali menší rozsah dat. Při porovnání s dekompozicí neupravené časové řady na grafu 4.18 zde můžeme vidět podobný trend i rezidua, ale velký rozdíl nastává v sezónní složce. Sezónna má zde délku sedm dní a reaguje na nulové hodnoty o víkendech. Při pohledu na predikce (viz graf 4.23) vidíme jednoznačné rozdíly. Pokud si odmyslíme víkendy, v pracovních dnech vidíme podobné hodnoty.

4.5.2 Automatizace

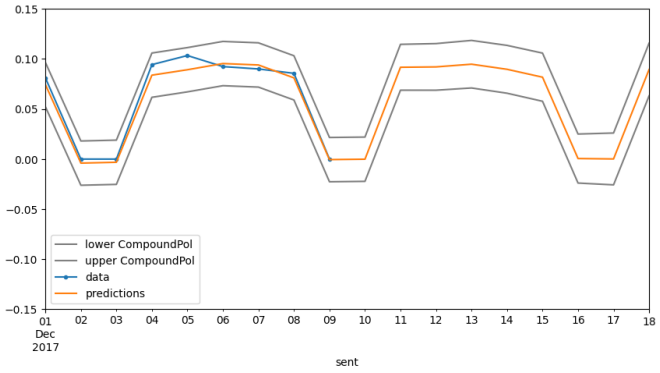
Modelování časových řad je většinu času manuální disciplína. Při automatizaci můžeme narazit na problémy a budeme muset také přijmout nějaká omezení a nečekat stejně kvalitní výsledky, jako při ruční analýze. V mnoha případech se musí přijmout nějaký kompromis. V našem případě to bude kvůli výkonu, konkrétně rychlosti. Například pokud uvažujeme sezónnost, může hledání vhodného modelu zabrat násobně mnoho času (zkoušíme více parametrů). Bude určitě záležet na datech a to jak na počtu, tak chování časové řady. V kódu 4.5 můžeme vidět porovnání doby běhu funkce na hledání optimálního modelu s parametrem sezónnosti nastaveného na *True* resp. *False*.

Výpis kódu 4.5 Porovnání času při sezónnosti a bez

```
best_model = pmdarima.auto_arima(**params,
                                 seasonal=True)

>Best model:  ARIMA(4,1,3)(3,0,2)[7]
>Total fit time: 491.388 seconds

best_model = pmdarima.auto_arima(**params,
                                 seasonal=False)
```



(a) Originální řada



(b) Upravená řada

Obrázek 4.23 Predikce budoucích hodnot na obou časových řadách

```
>Best model:  ARIMA(2,1,1)(0,0,0)[0]
>Total fit time: 5.367 seconds
```

V našem případě, kdy analýzu, získání trendu a predikce děláme na vyžádání, nemůžeme strávit tolik času pouze na hledání nejlepšího modelu. Budeme muset tedy udělat kompromis, prohledávat menší prostor parametrů, na úkor kvality predikce.

4.6 Implementace

Integraci do komunikační aplikace Slack budeme řešit jako Slack bota. Bot bude reagovat na definované příkazy uživatelů, kteří budou moct specifikovat i různé parametry. Bot bude implementován jako Flask aplikace, která jednoduše bude přesměrovávat requesty z určité URL a volat funkce naší aplikace. V těchto funkcích pak budeme využívat metody SlackAPI ke stahování a odesílání zpráv, které pak vhodně transformujeme a využijeme náš model pro zjištění nálady. Následně provedeme analýzu časové řady a tím získáme trend a predikce. Tyto všechny údaje zobrazíme v grafu a pošleme zpátky přes SlackAPI.

4.6.1 Slash commands

Jak již bylo řečeno, bot bude reagovat na definované příkazy. Slack tyto příkazy nazývá *slash commands*, protože mají formát */command*. V nastavení Slack bota musíme u jednotlivých přílazů nastavit adresu URL, kam Slack bude posílat data příkazu. Náš bot, Flask aplikace, bude na těchto adresách bude poslouchat a přesměrovávat dotazy na funkce. Příkazům pak můžeme přidat argumenty, které si ale sami musíme naparsovat, jelikož Slack posílá argumenty jako jeden řetězec. Argumenty pro všechny příkazy pak jsou:

- user – *string* – Emailová adresa uživatele, kterého budeme chtít analyzovat.
- date_from – *DD/MM/YYYY* – Datum, od kterého budeme provádět analýzu
- date_to – *DD/MM/YYYY* – Datum, do kterého budeme provádět analýzu.
- channel – *string* – ID kanálu, který chceme analyzovat. Bez tohoto parametru se bude analyzovat kanál, ve kterém příkaz použijeme.

Některé příkazy povinně vyžadují některé paramatry, ale omezení na datum není povinné nikde. Nyní si popíšeme jednotlivé příkazy.

Analyze

Nejobecnější příkaz, který nemá žádné povinné parametry. Provede analýzu kanálu, ze kterého příkaz spouštíme nebo kanálu specifikovaného v parametrech. Jedná se spíše o testovací příkaz, který se do praxe moc nehodí, jelikož většinou nechceme určitý kanál zaplnit grafy o náladě.

Analyze channel

Stejný příkaz jako */analyze*, jen je používaný pro analýzu jiného kanálu. Parametr *channel* je zde tudíž povinný.

User analysis

Opět se jedná o stejnou funkcionality, ale zde budeme filtrovat zprávy na základě uživatele. Zde je tedy poviný parametr *user*. Jelikož v analýze budou data jen od jednoho uživatele, s velkou pravděpodobností jich bude málo na extrakci trendu a predikce do budoucna. Na to musíme při použití tohoto příkazu (vlastně jakéhokoliv) myslet a používat ho rozumně.

Daily report

Tento příkaz označí kanál pro denní report nálady. Každý den v 8 hodin ráno se dělá analýza takto označených kanálů a chování je úplně stejné, jako při použití příkazu */analyze*. Při opětovném použití v daném kanálu se tento denní reporting zruší.

Leaderboard

Opět se provede analýza zpráv v kanálu, ale zde se ohodnocení nálady seskupuje podle uživatele. Do kanálu se pak vypíše žebříček uživatelé podle sentimentu jejich zpráv. Při velkém počtu uživatelů se vypíše jen tři „nejlepší“ a tři „nejhorší“.

Ukázku implemetace příkazu */analyze* můžeme vidět na kódu 4.6

Výpis kódu 4.6 Ukázka funkce reprezentující příkaz /analyze

```
@app.route('/analyze', methods=['POST'])
def analyze():
    data = request.form
    channel_id = data.get('channel_id')
    args_text = data.get('text')
    return channel_analysis(channel_id,
                            parse_args(args_text, client))
```

4.6.2 Funkce channel_analysis

Nyní si popíšeme funkci, která je jádrem celého procesu. Použití funkce jsme mohli už vidět v kódu 4.6, ale používá se i v dalších funkcích reprezentující jednotlivé příkazy. Jako argumenty funkci předáváme ID kanálu, který bude analyzován a už parsované argumenty ve formě slovníku. Funkce pak vrací třídu *Response*, která reprezentuje odpoveď *Flasku*, a číslo, vyjadřující typ odpovědi. Používají se tříciferná čísla, kde čísla začínající 2 znamenají úspěch, čísla začínající 5 značí chybu na straně serveru a čísla začínající 4 znamenají chybu na straně klienta.

Funkce si po zavolání uloží do proměnných potřebné parametry a případně zvolí výchozí, pokud některý chybí.

Výpis kódu 4.7 Hlavička funkce channel_analysis

```
def channel_analysis(channel_id: str, args: {},
                     output_channel=None) -> Response:
    ...
    ...
    ...
    return Response(), 200
```

To se stane například pokud se nespecifikuje parametr *channel*, který značí ID kanálu k analýze, kde se jako výchozí uvažuje kanál, ze kterého se příkaz poslal. Následně se pak načte historie zpráv z daného kanálu. Na to se využívá funkce Slack API, konkrétně pak funkci *conversations_history*. Limitace této funkce je načítání maximálně 200 zpráv najednou. Kvůli tomu musíme použít stránkování, kde musíme využít metadata odpovědí a jehož použití můžeme vidět v kódu 4.8. Jistě si zde můžeme všimnout uspání na půl sekundy, které je doporučené od vývojářů Slack API, k předejití zablokování kvůli mnoho žádostem.

Výpis kódu 4.8 Stránkování ve funkci načítající historii zpráv

```
response = client.conversations_history(channel=channel_id,
                                        limit=200)
history = response["messages"]
    while response['has_more']:
        time.sleep(.5)
        response =
        client.conversations_history(channel=channel_id,  limit=200,
               cursor = response['response_metadata']['next_cursor'])
        history = history + response["messages"]
```

Po načtení zpráv je musíme vyfiltrovat. Pokud jsou definované parametry *user*, *date_from* nebo *date_to*, zde budeme na jejich základě filtrovat. Také budeme ignorovat zprávy se stejným ID uživatele, jako máš náš bot. Po filtraci musíme ještě extra načíst vlákna, která se načítají přes extra funkci *conversations_replies*, a reakce. Nakonec všechny získané informace načteme do vlastní třídy, protože doteď byly uložené jako řetězce ve formátu json, viz 4.9

Výpis kódu 4.9 Zpráva načtená ze Slack API

```
{
    "type": "message",
    "user": "U1235837",
    "text": "Text\_zpravy",
    "ts": "123457689.000123"
}
```

Po filtraci proběhne analýza sentimentu. V kapitole jsme našli ideální model, logistickou regresi, a pro transformaci textu na příznaky použijeme metodu TF-IDF. Oba tyto modely potřebují natrénovat, což může trvat netriviální čas. Proto oba modely jsme uložili do souboru a při spuštění bota je ze souboru načteme, vše pomocí balíčku *pickle*. Kroky v analýze jsme už

popisovali v předchozích kapitolách (očištění, stemming), zde ale navíc musíme počítat s dalším problémem. Jelikož modely máme natrénovány na anglických datech, musíme analyzovat taktéž anglické zprávy. Zprávy ve skutečnosti nebudou jen v angličtine, ale můžou být v jakémkoliv jazyku, případně se i jazyky můžou míchat. Proto musíme zprávy přeložit a na to využijeme Google Translator API. Z hotové analýzy poté můžeme extrahovat trend a učinit predikce, které budou vždy na deset dní dopředu. Všechny tři výsledky pak vhodně zanaseme do grafů a pošleme zpátky přes Slack API. Během celého běhu funkce může nastat několik problémů. V případě problémů s predikcí nebo trendem se pošle jen prázdný graf s chybovou hláškou. V případě jiného problému (např. s API) se odešle *Response* s kódem 500.

Kapitola 5

Testování

Jedním z úkolů této práce bylo otestovat chod celé aplikace na reálných datech z firemního prostředí. V této kapitole si tedy představíme firmu, ve které testování proběhne, popíšeme nasazení aplikace a problémy a ukážeme výsledky testování.

5.1 O firmě Ackee

Firma začala jako startup v inkubátoru ČVUT a zabývá se designováním a vývojem mobilních i webových aplikací, kdy má za sebou přes 300 úspěšných projektů [37]. Sídlí v Berlíně a Praze. Mimo zařazení v roce 2020 mezi 500 nejrychleji rostoucích technologických firem v regionu EMEA firma získala další ocenění například od časopisu Forbes, E15.cz, Delloite a také vyhrála mezinárodní soutěž German Brand Award pořádanou prestižním German Design Councilem [37].

5.2 Nasazení

Pro lokální spuštění stačilo aplikaci spustit a využít k tomu navíc nástroj *ngrok*[1], který přesměrovává requesty z veřejné IP adresy na lokální. Pro testování aplikace ve firemní prostředí musíme udělat pár věcí. Pro přístup k metodám Slack API jsou potřeba dva tokeny, signing secret a OAuth token, které jsou pro každý Slack workspace unikátní. Pro zachování testovacího prostředí tedy aplikaci nahrajeme na Oracle Cloud [2], kde je možné hostovat jednoduché aplikace na omezených virtuálních strojích zdarma. Musíme i vytvořit novou aplikaci na Slacku. Pro jednoduchou konfiguraci a kvůli faktu, že nemáme na firemní Slack přístup, vytvoříme konfigurační soubor (viz příloha A), díky kterému lze vytvořit aplikaci instantně a bez dalšího nastavování. Poté již stačí získat zmíněné tokeny a uložit je do aplikace.

[1] ngrok, https://ngrok.com/
[2] Oracle Cloud, https://www.oracle.com/cloud/free/

5.2.1 Problémy

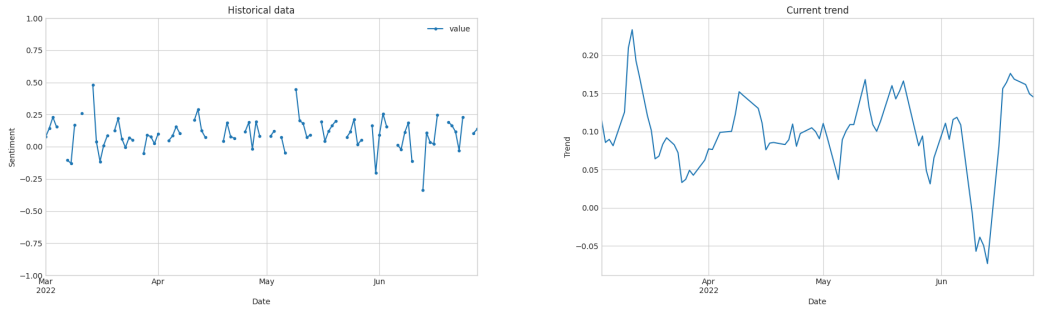
Při testování aplikace nastalo několik problémů, které bylo potřeba vyřešit. V první řadě se jednalo o zprávy bez odesílatele. Tato situace nastala, pokud autor zprávy již nebyl členem určeného workspacu. Původní implementace s tím nepočítala, ale jednalo se o rychlou úpravu. Zprávy bez odesílatele se normálně zpracovávaly, jen se ignorovaly v případě žebříčku.

V druhé řadě se v grafech historie nálady ukazovaly nulové hodnoty. Nebylo tak zřejmé, jestli se jedná o chybějící data nebo neutrální náladu. Ani zde oprava nebyla nijak složitá.

Jedním z (prakticky) neřešitelných problémů je časová náročnost celé operace. Při načítání velkého počtu zpráv může příkaz trvat i přes několik desítek minut. Jistě by se celá aplikace dala více optimalizovat, ale největší časové zdržení je kvůli Slack API. Každý uživatel API má dovolený určitý počet žádostí, který když překročí, přestane Slack API pro něj na chvíli fungovat. Vývojáři proto doporučují při načítání zpráv vždy volat funkci po uspání programu na jednu sekundu. A jelikož zpráv lze najednou načíst jen 200 a zprávy z vláken se zpracovávají extra, můžeme vidět, že při velkém počtu zpráv zde dojde k velkému zdržení.

5.3 Testování

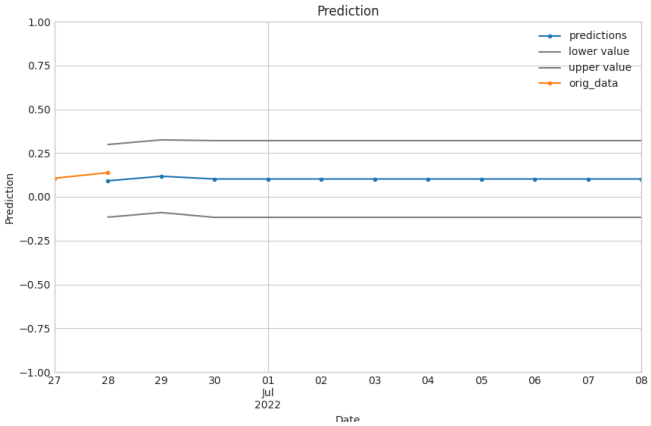
Na následujících grafech můžeme vidět výsledky z jednotlivých testování. K analýze byl použit příkaz */analyze_channel*. Celkový počet zpráv se pohybuje v rozmezí $1000 - 5000$. V historii na grafech 5.1, 5.3 a 5.5 si můžeme všimnout relativně velkého množství chybějících dat. Vidíme zde i celkovou náladu spíše pozitivního rázu. Jelikož se skóre nálady za každý den průměruje, nevidíme zde skoro žádné extrémy. Kvůli chybějícím datům na konci sledovaného období, si pak můžeme všimnout relativně nezajímavým predikcím na grafech 5.2, 5.4 a 5.6. Tyto predikce se pohybují kolem 0, což je prostředek intervalu hodnot nálady.



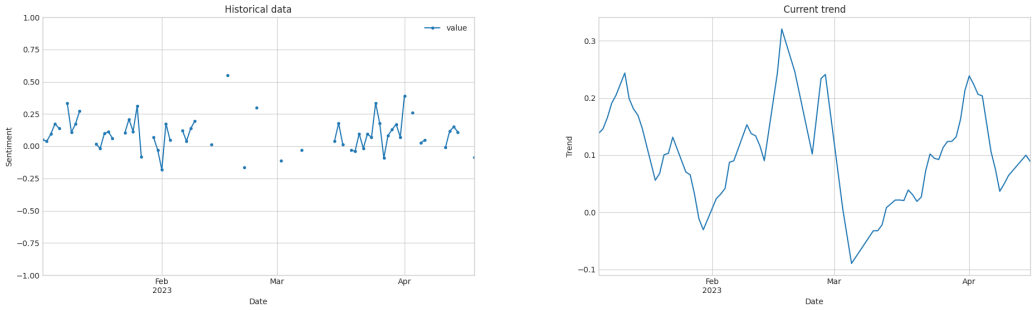
Obrázek 5.1 Historická data a trend 1.

5.3.1 Závěr

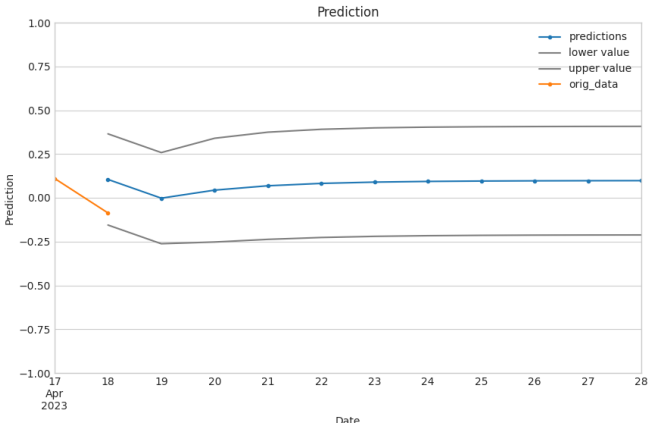
Firma Ackee dodala pozitivní zpětnou vazbu a vyjádřila spokojenost s prototypem aplikace. Na závěr dodala vidinu možného příslibu do budoucna na další rozšíření pro každodenní používání za účelem určování kvality projektu, konkrétně metrikou zjišťování spokojenosti lidí na projektech.



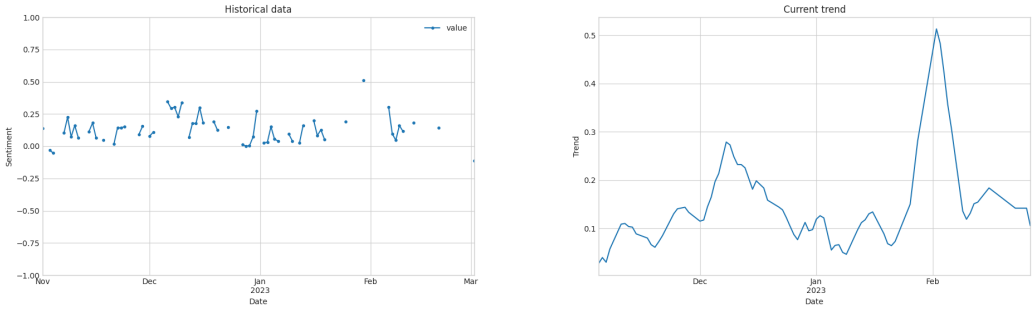
Obrázek 5.2 Predikce 1



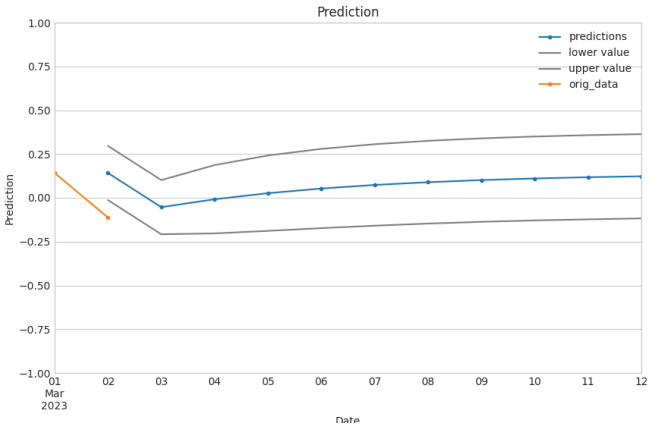
Obrázek 5.3 Historická data a trend 2.



Obrázek 5.4 Predikce 2



Obrázek 5.5 Historická data a trend 3.



Obrázek 5.6 Predikce 3

Kapitola 6

Závěr

V této diplomové práci jsme se zabývali analýzou nálady v komunikačních aplikacích. Úkolem této práce bylo vytvořit nástroj integrovaný do komunikační platformy. Pro tento účel jsme analyzovali aktuální používané metody a nástroje pro určení sentimentu a navrhli vlastní metriky pro měření vývoje nálady, které braly v potaz různé formy vyjádření sentimentu. Pro predikce bylo na vývoj nálady pohlíženo jako na časovou řadu, které mají aparát na predikování budoucí hodnoty a extrakci trendu. Tento nástroj byl implementován ve formě bota na komunikační platformě Slack. Celé řešení je dostupné jako open-source projekt na platformě GitHub. Implementace proběhla v jazyce Python a samotný bot je implementovám pomocí balíčku Flask.

Bot reaguje na definované příkazy, které lze modifikovat různými parametry. Po analýze pošle bot zpátky tři grafy s historií nálady, trendem a predikcí do budoucna. Časově na tom analýza není moc dobře, ale jedná se spíše o limitaci ze strany SlackAPI, které nelze obejít.

Při hledání vhodného modelu jsme testovali různé druhy s různými parametry a nejúspěšnější byl model logistické regrese. Netestovali jsme jen model, ale i různé metody zpracování textu, například metody transformace textu na příznaky, a předzpracování dat, kde jsme se zaměřili na balancování dat. V rámci práce proběhla i rešerše existujících řešení. Proto jsme pak výsledný model a nejlepší techniky zpracování textu porovnávali s některými těmito řešeními a poskytl relativně srovnatelné výsledky.

V budoucnosti by bylo možné rozšířit implementaci o více konfigurovatelné příkazy. Uživatel by si například mohl zvolit specifický model nebo nastavovat jednotlivé parametry. Také by se dalo přemýšlet o lepší predikci budoucích hodnot, kde často docházelo k neuspokojenosti z důvodu malého počtu dat.

Příloha A

Manifest.yaml

```
display_information:
  name: Sentiment Analysis Bot
features:
  bot_user:
    display_name: Sentiment Analysis Bot
    always_online: false
  slash_commands:
    - command: /analyze
      url: http://xxx.xxx.xxx.xxx:5000/analyze
      description: Analyze this channel
      should_escape: false
    - command: /leaderboard
      url: http://xxx.xxx.xxx.xxx:5000/leaderboard
      description: Get leaderboard based on SA
      should_escape: false
    - command: /user_analysis
      url: http://xxx.xxx.xxx.xxx:5000/user_analysis
      description: Analyze sentiment of specified user
      should_escape: false
    - command: /analyze_channel
      url: http://xxx.xxx.xxx.xxx:5000/analyze_channel
      description: Analyze specified channel
      should_escape: false
oauth_config:
  scopes:
    bot:
      - channels:history
      - channels:read
      - chat:write
      - commands
```

```
      - files:write
      - users:read.email
      - users:read
settings:
  org_deploy_enabled: false
  socket_mode_enabled: false
  token_rotation_enabled: false
```

Bibliografie

1. MEDHAT, Walaa; HASSAN, Ahmed; KORASHY, Hoda. *Sentiment analysis algorithms and applications: A survey.* Sv. 5 [online]. 2014. [cit. 2023-05-02]. Č. 4. ISSN 2090-4479. Dostupné z DOI: https://doi.org/10.1016/j.asej.2014.04.011.
2. TSYTSARAU, Mikalai; PALPANAS, Themis. *Survey on mining subjective data on the web.* Sv. 24 [online]. 2012. [cit. 2023-05-02]. Dostupné z DOI: https://doi.org/10.1007/s10618-011-0238-6.
3. FELDMAN, Ronen. Techniques and Applications for Sentiment Analysis. *Commun. ACM* [online]. 2013, roč. 56, č. 4, s. 82–89 [cit. 2023-05-02]. ISSN 0001-0782. Dostupné z DOI: 10.1145/2436256.2436274.
4. NARAYANAN, Ramanathan; LIU, Bing; CHOUDHARY, Alok. Sentiment analysis of conditional sentences. In: *Proceedings of the 2009 conference on empirical methods in natural language processing* [online]. 2009, s. 180–189 [cit. 2023-05-02].
5. JINDAL, Nitin; LIU, Bing. Identifying Comparative Sentences in Text Documents. In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* [online]. Seattle, Washington, USA: Association for Computing Machinery, 2006, s. 244–251 [cit. 2023-05-02]. SIGIR '06. ISBN 1595933697. Dostupné z DOI: 10.1145/1148170.1148215.
6. SCHUETZE, Hinrich. *Stemming and lemmatization* [online]. Stanford.edu, 2009. [cit. 2023-05-02]. Dostupné z: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html.
7. AKUMA, Stephen; LUBEM, Tyosar; ADOM, Isaac Terngu. Comparing Bag of Words and TF-IDF with different models for hate speech detection from live tweets. *International Journal of Information Technology* [online]. 2022, s. 1–7 [cit. 2023-05-02].
8. KLOUDA, Karel; VAŠATA, Daniel. *FIT ČVUT: Předmět Machine learning 1* [online]. 2023. [cit. 2023-04-10]. Dostupné z: https://courses.fit.cvut.cz/BI-ML1/lectures.
9. AYODELE, Taiwo Oladipupo. Types of machine learning algorithms. *New advances in machine learning* [online]. 2010, roč. 3, s. 19–48 [cit. 2023-05-02].

10. KLOUDA, Karel; VAŠATA, Daniel. *FIT ČVUT: Předmět Machine learning 1* [online]. 2023. [cit. 2023-04-10]. Dostupné z: https://courses.fit.cvut.cz/BI-ML1/lectures/files/BI-ML1-07-cs-handout.pdf.

11. IBM. *What is the k-nearest neighbors algorithm? — IBM* [online]. www.ibm.com, 2023. [cit. 2023-04-10]. Dostupné z: https://www.ibm.com/topics/knn.

12. KOTSIANTIS, S. B. Decision trees: a recent overview. *Artificial Intelligence Review* [online]. 2013, roč. 39, č. 4, s. 261–283 [cit. 2023-05-02]. ISSN 1573-7462. Dostupné z DOI: 10.1007/s10462-011-9272-4.

13. RUSSELL, Stuart J.; NORVIG, Peter. *Artificial Intelligence: A Modern Approac* [online]. 3. vyd. Upper Saddle River: Prentice Hall, 2010 [cit. 2023-05-02]. ISBN 0-13-604259-7.

14. *API Reference — scikit-learn 0.20.3 documentation* [online]. Scikit-learn.org, 2017. [cit. 2023-05-02]. Dostupné z: https://scikit-learn.org/stable/modules/classes.html.

15. MURPHY, Kevin P et al. Naive bayes classifiers. *University of British Columbia* [online]. 2006, roč. 18, č. 60, s. 1–8 [cit. 2023-05-02].

16. BERTSEKAS, Dimitri; TSITSIKLIS, John N. *Introduction to probability*. Sv. 1 [online]. Athena Scientific, 2008 [cit. 2023-05-02].

17. SCIKIT-LEARN. *1.9. Naive Bayes — scikit-learn 0.21.3 documentation* [online]. Scikit-learn.org, 2019. [cit. 2023-05-02]. Dostupné z: https://scikit-learn.org/stable/modules/naive_bayes.html.

18. *1.4. Support Vector Machines* [online]. scikit-learn, [b.r.]. [cit. 2023-05-02]. Dostupné z: https://scikit-learn.org/stable/modules/svm.html.

19. HEARST, M.A.; DUMAIS, S.T.; OSUNA, E.; PLATT, J.; SCHOLKOPF, B. Support vector machines. *IEEE Intelligent Systems and their Applications*. 1998, roč. 13, č. 4, s. 18–28. Dostupné z DOI: 10.1109/5254.708428.

20. KANAL, Laveen N. Perceptron. In: *Encyclopedia of Computer Science* [online]. GBR: John Wiley a Sons Ltd., 2003, s. 1383–1385 [cit. 2023-05-02]. ISBN 0470864125.

21. . *1.17. Neural network models (supervised)* [online]. Scikit-learn.org, 2019. [cit. 2023-05-02]. Dostupné z: https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

22. DOMINGUES, Ines; AMORIM, José P.; ABREU, Pedro H.; DUARTE, Hugo; SANTOS, João. Evaluation of Oversampling Data Balancing Techniques in the Context of Ordinal Classification. In: *2018 International Joint Conference on Neural Networks (IJCNN)* [online]. 2018, s. 1–8 [cit. 2023-05-02]. Dostupné z DOI: 10.1109/IJCNN.2018.8489599.

23. *What Is Undersampling?* [online]. Master's in Data Science, [b.r.]. [cit. 2023-05-02]. Dostupné z: https://www.mastersindatascience.org/learning/statistics-data-science/undersampling/.

24. . *3. Model selection and evaluation* [online]. Scikit-learn.org, 2019. [cit. 2023-05-02]. Dostupné z: https://scikit-learn.org/stable/model_selection.html.

25. HAMILTON, James Douglas. *Time series analysis* [online]. Princeton university press, 2020 [cit. 2023-05-02].

26. HYNDMAN, R.J.; ATHANASOPOULOS, G. *Forecasting: principles and practice: 3rd edition* [online]. 3. vyd. Melbourne, Australia: OTexts, 2021 [cit. 2023-05-02].

27. DEDECIUS, Kamil. *FIT ČVUT: Předmět Statistická analýza časových řad* [online]. 2023. [cit. 2023-05-02]. Dostupné z: `https://courses.fit.cvut.cz/NI-SCR/lectures/index.html`.

28. CRYER, J. D.; CHAN, K.-S. *Time series analysis: second edition*. 2. vyd. New York: Springer-Verlag, 2008. ISBN 978-0-387-75958-6.

29. CHATFIELD, Christopher. *The analysis of the time series: an introduction* [online]. 6th ed. Boca Raton: Chapman & Hall/CRC, 2004 [cit. 2023-05-02]. ISBN 9781584883173.

30. *NLTK :: Natural Langue Toolkit* [online]. www.nltk.org, [b.r.]. [cit. 2023-04-28]. Dostupné z: `https://www.nltk.org/index.html`.

31. TEXTBLOB. *TextBlob: Simplified Text Processing — TextBlob 0.15.2 documentation* [online]. Readthedocs.io, 2018. [cit. 2023-05-02]. Dostupné z: `https://textblob.readthedocs.io/en/dev/`.

32. MONKEYLEARN. *14 of The Best Sentiment Analysis Tools* [online]. MonkeyLearn Blog, 2019. [cit. 2023-05-02]. Dostupné z: `https://monkeylearn.com/blog/sentiment-analysis-tools/`.

33. SLACK. *Features of Slack* [online]. Slack, 2017. [cit. 2023-04-16]. Dostupné z: `https://slack.com/features`.

34. SLACK. *Slack API Documentation* [online]. Slack API, [b.r.]. [cit. 2023-04-16]. Dostupné z: `https://api.slack.com/docs`.

35. KAGGLE. *Kaggle: Your Home for Data Science* [online]. Kaggle.com, 2022. [cit. 2023-05-02]. Dostupné z: `https://www.kaggle.com/`.

36. *statsmodels.tsa.arima.model.ARIMAResults.plot_diagnostics — statsmodels* [online]. www.statsmodels.org, [b.r.]. [cit. 2023-05-02]. Dostupné z: `https://www.statsmodels.org/dev/generated/statsmodels.tsa.arima.model.ARIMAResults.plot_diagnostics.html`.

37. *Ackee - Agentura pro vývoj mobilních a webových aplikací* [online]. Ackee, [b.r.]. [cit. 2023-05-02]. Dostupné z: `https://www.ackee.cz/`.

Obsah přiloženého média

```
README.MD ..... stručný popis implementace a použití
src ..... zdrojové kódy implementace
Notebooks ..... Jupyter notebooky, používané při implementaci
doc ..... zdrojové soubory textové části
    thesis.pdf ..... text práce ve formátu PDF
```