Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Detekce kategorie obsahu webové stránky prostřednictvím metod strojového učení

Bakalářská práce

Patrik Dohnal

Školitel: Ing. Jan Fesl, Ph.D.

České Budějovice 2021

Jihočeské univerza v Českých Budějovicích

Přírodovědecká fakulta

ZADÁVACÍ PROTOKOL BAKALÁŘSKÉ PRÁCE

Student: Patrik Dohnal
(jméno, příjmení, tituly)

Obor - zaměření studia: Aplikovaná Informatika

Katedra: Ústav aplikované informatiky

Školitel: Ing. Jan Fesl, Ph.D.
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, email)

Garant z Přírodovědecké fakulty: .....
(jméno, příjmení, tituly, katedra - jen v případě externího školitele)

Školitel - specialista, konzultant:
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, email)

Téma bakalářské práce:
Detekce kategorie obsahu webové stránky prostřednictvím metod strojového učení

Popis práce:

Automatická klasifikace obsahu webových stránek je dnes velmi žádanou disciplínou, která má vícečetné využití – a sice, pro fulltextové vyhledávače umožňuje zařazení webových stránek do konkrétních kategorií pro budoucí indexování, z hlediska bezpečnosti umožňuje např. odhalení stránek s tematicky závadným obsahem atd.

Z hlediska pochopení obsahu stránky je důležitá syntéza textové a vizuální informace, která je pro výsledný dojem stěžejní. Pro klasifikaci obsahu stránky budou použity ověřené klasické metody popř. moderní metody strojového učení, které jsou pro dané zaměření vhodné (zejména z oblasti deep-learningu, např. sítě pro zpracování obrázků a zpracování přirozeného jazyka).

Cíle práce:

1) Rešerše v oblasti metod pro automatickou klasifikaci webových stránek, klasické metody a nové možné přístupy, včetně detailního popisu.
2) Vytvoření data-setu pro verifikaci a odladění použitých metod (např. pro tvorbu modelu neurální sítě)
3) Vytvoření systému klasifikace obsahu webových stránek – pravděpodobně hierarchického se stromovou strukturou
4) Implementace výsledného řešení v jazyku C++ či Python
5) Ověření navrženého konceptu a případném odůvodnění nedostatků

Základní doporučená literatura:

[1] F. Chollet, Deep learning with Python, Manning Publications, ISBN: 9781617294433, 2017

[2] RNDr. Jiří Materna, Ph.D., Automatické určení domény a klíčových slov stránky, diplomová práce FI MUNI, 2008

Financování práce: .....
Vedoucí práce: Ing. Jan Fesl, Ph.D. podpis: .....
U externích vedoucích fakultní garant práce: podpis: .....
Garant oboru bak. studia (nepožaduje se u zaměření, příprava na mag. stud. biologie):
..... podpis: .....
Vedoucí katedry: Rudolf Vohnout podpis: .....
Odborný konzultant (chemická část): podpis: .....
Případný souhlas vedoucí ústavu AV: ..... podpis: .....

V Českých Budějovicích dne: ....10.12.2019.....

Převzal/a dne: ....29. 1. 2020..... podpis: .....

Bibliografické údaje

Dohnal, P., 2021: Detekce kategorie obsahu webové stránky prostřednictvím metod strojového učení. [Detection of websites content category using machine learning methods. Bc. Thesis, in Czech.] – 44 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Tato bakalářská práce se zabývá návrhem systému pro klasifikaci obsahu webových stránek včetně následné implementace v programovacím jazyce Python. K samotné klasifikaci jsou využívány modely strojového učení jako jsou Naivní Bayesův klasifikátor, K-Nejbližších sousedů a Support Vector Machines. V rámci celého procesu se rovněž předpokládá tvorba vlastní množiny dat, na kterých jsou tyto modely trénovány a následně testovány. Součástí práce je i podrobná rešerše použitých metod.

Klíčová slova

Strojové učení, Text mining, Klasifikace, Dataset, Webové stránky, Předzpracování dat, Zakódování textu, TF-IDF, Support Vector Machines, Naivní Bayesuv klasifikátor, K-Nejbližší sousedů

Annotation

This bachelor thesis is focused on design and the implementation of the algorithm for classifying the websites into a several categories. The implementation of this software is written in Python. For classifying purposes I use machine learning models such as Naive Bayes classifier, K-Nearest neighbors and Support Vector Machines. Within the process it is assumed to collect my own dataset, wich will be used for training and testing purposes. Thesis also includes detailed description of the methods I uesd.

Keywords

Machine learning, Text Mining, Classification, Dataset, Webpages, Data preprocessing, Word embedding, TF-IDF, Support Vector Machines, Naive Bayes classifier, K-Nearest neighbors

Prohlášení

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne: 29. 4. 2021 Podpis autora: Patrik Dohnal

Poděkování

Tímto bych rád poděkoval svému vedoucímu Ing. Janu Feslovi, Ph.D. za skvělou spolupráci, trpělivost, jeho čas strávený při konzultacích a za odbornost poskytnutou při tvorbě tohoto experimentálního softwaru a psaní této práce.

Obsah

1 Úvod

Tato práce se zabývá návrhem a implementací experimentálního softwaru pro automatickou klasifikaci webových domén na základě textových informacích, které jsou obsaženy na dané webové stránce. K tomuto procesu je využíváno metod strojového učení a dalších postupů z text miningového odvětví. Automatická klasifikace je dnes velmi žádanou disciplínou, která našla mnohočetné využití v řadě různých oborů.

Text mining a zpracování přirozeného jazyka se dnes běžně využívá při fulltextovém vyhledávání, indexaci či automatickou tvorbu katalogů. Nachází také uplatnění při našeptávání následujících slov, a to jak ve větách, tak i například při psaní kódů v pokročilejším vývojovém prostředí programátora. Zejména automatická klasifikace by mohla v budoucnu poskytnout různé bezpečnostní výhody, a to konkrétně pro snazší detekci závadného obsahu, na který by bylo možné patřičně reagovat. Takový software by mohl být vítán jak v podnikovém, tak i v menším rodinném prostřední.

Návrh softwaru pro automatické členění webových stránek do kategorií vyžaduje několik patřičných kroků, kterými se tato práce podrobně zabývá. V první řadě je nutné si opatřit množinu webových stránek, ze kterých bude následně stahován veškerý text, k tomuto účelu jsou využity techniky web scrapingu. Z textu se později, po několika dalších krocích, sestaví takzvaný dataset, tedy množina dat, která slouží pro trénování a testování klasifikačních modelů. V této práci jsou pro klasifikaci dat využity tři různé modely – Naivní Bayesuv klasifikátor, K-Nejbližších sousedů a Support Vector Machines. Implementace softwaru je psána v programovacím jazyce Python.

2 Cíl

Tato bakalářská práce se zabývá klasifikací webových stránek prostřednictvím metod strojového učení. Z tohoto důvodu je nutné provést literární rešerši dostupných metod z oblasti strojového učení a rovněž z oblasti text miningu. Detailní popis metod bude obsažen v teoretické části této bakalářské práce. Následně má být navrhnut a implementován experimentální software, který ověří funkčnost těchto metod pro úlohu klasifikace webových stránek. Pro implementaci takového algoritmu bude nejprve nutné vytvořit množinu, ve které budou obsaženy webové stránky několika kategorií. Tato množina stránek bude sloužit k naučení a otestování klasifikačních modelů. Po procesu učení by měl být tento software schopen kategorizovat neznámé webové stránky, které se objeví na vstupu programu. Pro implementaci softwaru má být použit programovací jazyk C++ nebo Python.

Konkrétní cíle:

1) Rešerše v oblasti metod pro automatickou klasifikaci webových stránek, klasické metody a nové možné přístupy, včetně detailního popisu
2) Vytvoření datasetu pro verifikaci a odladění použitých metod
3) Vytvoření systému klasifikace obsahu webových stránek
4) Implementace výsledného řešení v programovacím jazyku C++ či Python
5) Ověření navrženého konceptu a případné odůvodnění nedostatků

3 Teoretická část

3.1 Klasifikace

Tato disciplína je z hlediska strojového učení synonymem pro třídění či kategorizaci. Klasifikační úloha je proces přiřazení kategorie vstupním datům na výstupu. V případě použití následujících metod se jedná o proces učení s učitelem, kde na předem známých trénovacích datech je přesně dáno, do které kategorie patří konkrétní vstup z trénovací množiny.

Jednomu záznamu z trénovací množiny říkáme vzor nebo také instance či záznam. Množinu všech vzorů lze poté definovat jako X, kde $X = \{x_1, x_2, x_3, \dots, x_{n,}\}$, přičemž $x_i = (f_1^{(i)}, f_2^{(i)} \dots, f_k^{(i)})$, $1 \leq i \leq n$,$n = |X|$, kterým říkáme *features*. Pro všechna $x_i \in X$ existuje zobrazení $X \to Y$, kde Y je množina kategorií pro klasifikaci daných vzorů. $Y = \{y_1, y_2, y_3, \dots, y_{m,}\}$, kde y_j, $1 \leq j \leq m$, $m = |Y|$.

Na základě tohoto učení jsou později klasifikační modely schopny kategorizovat neznámá data a realizovat tak funkci $f(x) = y$, kde $f: X \to Y$.

3.1.1 K-Nejbližších sousedů

Je jedna z nejjednodušších metod klasifikace, vychází z algoritmu hledání nejbližšího souseda, který je následující. Při klasifikaci neznámého vstupního vektoru se do paměti načte celá trénovací množina a hledá se vzor s nejmenší vzdáleností od právě klasifikovaného vektoru. Pro výpočet se využívá Eukleidovská vzdálenost. Nově klasifikovaný vektor zařadíme do stejné kategorie, jako je nejbližší vektor z trénovací množiny.

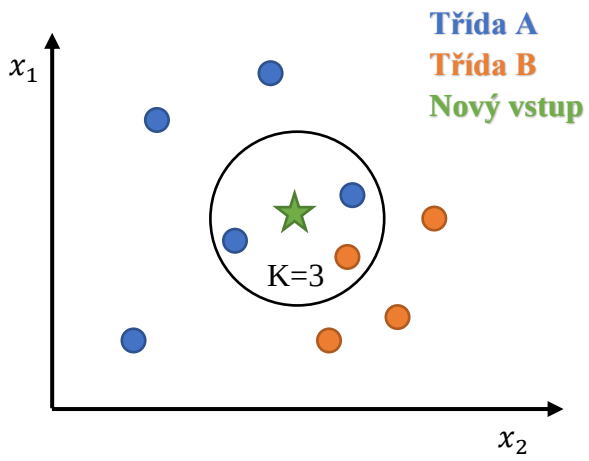
Pokud bychom uvažovali data reprezentovaná dvourozměrnými vektory, vzdálenost mezi nimi bychom spočítali jednoduše pomocí Pythagorovy věty. U klasifikační disciplíny ovšem běžně narážíme na vektory o daleko více dimenzích. Mějme dva třírozměrné vektory $A = (a_1, a_2, a_3)$ a $B = (b_1, b_2, b_3)$, Euklidovskou vzdálenost mezi těmito dvěma vektory vypočteme následujícím způsobem.

$$d(a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2} \quad (1)$$

Můžeme si všimnout, že výpočet je velmi podobný Pythagorově větě. Obecně pro N rozměrné vektory se Eukleidovská vzdálenost počítá následovně. Uvažujeme-li vektory $C = (c_1, \dots, c_n)$ a $D = (d_1, \dots, d_n)$, poté je jejich vzdálenost definována jako [1]

$$d(C, D) = \sqrt{(c_1 - d_1)^2 + \cdots + (c_n - d_n)^2} = \sqrt{\sum_{i=1}^{N} (c_i - d_i)^2} \qquad (2)$$

Výpočet metodou K-Nejbližších sousedů vychází ze stejného principu, v tomto případě ovšem nepovažujeme jako vítěze pouze nejbližší vektor z trénovací množiny nýbrž většinovou převahu z K nejbližších vektorů, kde K je parametr metody a zpravidla pro něj volíme liché číslo. Pokud bychom za K zvolili číslo sudé, mohlo by se stát, že nejbližší okolní vektory dvou různých tříd by byly zastoupeny v poměru 1:1 a bylo by tak nemožné rozhodnout, do které třídy bude nově klasifikovaný vstup patřit. [2]



Obrázek 1 – Klasifikace metodou K-Nejbližších sousedů

Výhodou této metody je v první řadě jednoduchost celého algoritmu a tím pádem i snadná implementace. Výpočetní náročnost závisí na množství vzorů z trénovací množiny a na velikosti vstupních vektorů

Hlavní nevýhodou k-NN je značná paměťová náročnost, při klasifikaci dat totiž musí být v paměti načtený trénovací dataset. S rostoucí trénovací množinou stoupá jak paměťová, tak i výpočetní náročnost.

3.1.2 Naivní Bayesuv Klasifikátor

Naivní Bayesuv klasifikátor je poměrně jednoduchý klasifikátor založený na statistických metodách. Vychází z Bayesovy věty, která se zabývá vztahem podmíněné pravděpodobnosti s opačnou podmíněnou pravděpodobností [3].

$$P(\,E|H\,) = \frac{P(\,H|E\,)\,P(\,H\,)}{P(\,E\,)} \tag{3}$$

Písmeno E zastupuje slovo evidence, písmeno H poté hypotézu. Pravděpodobnost hypotézy $P(H)$ se nazývá apriorní a vyjadřuje zastoupení jednotlivých hypotéz bez ohledu na další informace. Podmíněná pravděpodobnost $P(E|H)$ je nazývána aposteriorní a říká, jakým způsobem se bude měnit pravděpodobnost hypotézy za předpokladu, že nastalo H. Pravděpodobnost evidence ve vzorci zastupuje jmenovatel $P(E)$ [3].

Pokud je více hypotéz, mezi kterými se rozhodujeme, zajímá nás pro danou evidenci pouze ta nejpravděpodobnější. Mějme hypotézy $H_t, t = 1, \dots, T$, poté spočítáme $P(H_t|E)$ a vybereme z nich H_{MAP}, tedy hypotézu s největší aposteriorní pravděpodobností. Při výpočtu nás rovněž nezajímají konkrétní hodnoty, nýbrž která $H_t = H_{MAP}$, tudíž můžeme vynechat jmenovatele výpočtu. S touto úvahou platí [4]:

$$H_{MAP} = H_j \;\Leftrightarrow\; P\big(E\big|H_j\big)P\big(H_j\big) = max_t\big(P(E|H_t)\,P(H_t)\big) \tag{4}$$

Naivní Bayesuv klasifikátor předpokládá, že jednotlivé evidence jsou nezávislé při platnosti hypotézy H, tento fakt umožňuje výpočet aposteriorní pravděpodobnosti hypotézy při platnosti všech evidencí [4], proto se také nazývá *Naivní*. V praxi je ovšem velmi obtížné dosáhnout této nezávislosti, to vede k myšlence, zdali takovýto klasifikátor může reálně fungovat. V praxi se později ukázalo, že tato klasifikace funguje velmi dobře i na závislých datech.

$$P(H|E_1, \dots, E_k) = \frac{P(H)}{P(E_1, \dots, E_k)} \prod_{k=1}^{k} P(E_k|H)$$

V případě naší klasifikace webových stránek to znamená, že $P(H|E_1, \dots, E_k)$ vyjadřuje pravděpodobnost, že webová stránka patří do třídy H, pokud se na vstupu objevil vektor atributů $(f_1, \dots, f_k)$. Člen $P(H)$ reprezentuje pravděpodobnost výskytu třídy H, a to na základě předchozích zkušeností získaných při učení na trénovacích datech [5]. $P(E_k|H)$ vyjadřuje pravděpodobnost výskytu atributu E_k v třídě Y (na základě učení).

Z teorie o Bayesovské klasifikaci již víme, že jmenovatele můžeme zanedbat, pakliže budeme hledat třídu s nejvyšší aposteriorní pravděpodobností H_{MAP}.

$$H_{MAP} = H_j \Leftrightarrow P(H_j)\prod_{k=1}^{k} P(E_k|H_j) = max_t\left(P(H_t)\prod_{k=1}^{k} P(E_k|H_t)\right) \quad (6)$$

$$P(H_t) = \frac{n_t}{n}$$

$$P(E_k|H_t) = \frac{n_t(E_k)}{n_t}$$

Kde n_t je počet webových stránek s konkrétní třídou H_t a jmenovatel n je počet všech webových stránek. Člen $n_t(E_k)$ vyjadřuje počet případů, kdy je atribut E_k obsažený ve třídě H_t.

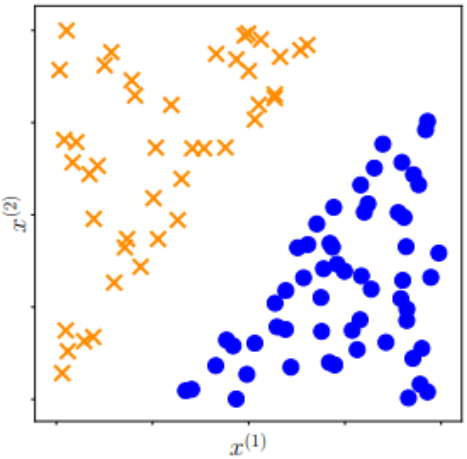
Nevýhodou tohoto přístupu je fakt, že pokud se během učení daný atribut nevyskytl ve třídě, tak při každé další klasifikaci pro tuto danou třídu s tímto konkrétním atributem, bude výsledek výpočtu roven 0 a není tedy bráno v potaz, jestli ostatní atributy nevykazovaly vyšší výsledky. Vstup klasifikátoru by tedy byl nesprávně zařazen do jiné třídy, protože výsledek pro správnou třídu by, na základě tohoto výpočtu, vyšel 0. Tato skutečnost se dá obejít například Laplaceovou korekcí [4]

$$P(E_k|H_t) = \frac{n_t(E_k) + 1}{n_t + T} \quad (7)$$

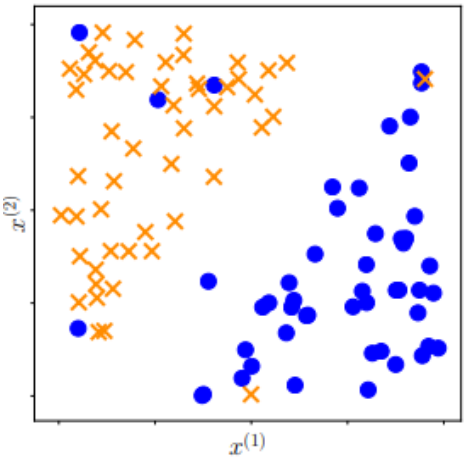
Kde T je počet tříd.

3.1.3 Support Vector Machines

Support Vector Machines (SVM) poslední dvě desetiletí velmi narůstá na popularitě, díky velmi dobrým výsledkům při klasifikaci více dimenzionálních dat. Klasifikace textu je přesným příkladem práce s daty o stovkách různých dimenzích, kde jednotlivé dimenze tvoří atributy vstupů do klasifikátoru, tzv. *features* (slova). Základem je lineární SVM, který separuje dvě odlišné třídy pomocí nadroviny. Jak již název napovídá, jedná se o metodu podpůrných vektorů, ty tvoří krajní body dvou odlišných tříd, úkolem SVM je najít takovou nadrovinu, která by co nejlépe rozdělovala dvě třídy, najít podpůrné vektory, které maximalizují vzdálenost mezi jednotlivou třídou a hraniční nadrovinou (maximum margin).



Obrázek 3 – Lineárně separovatelná data, obrázek převzat z [6]

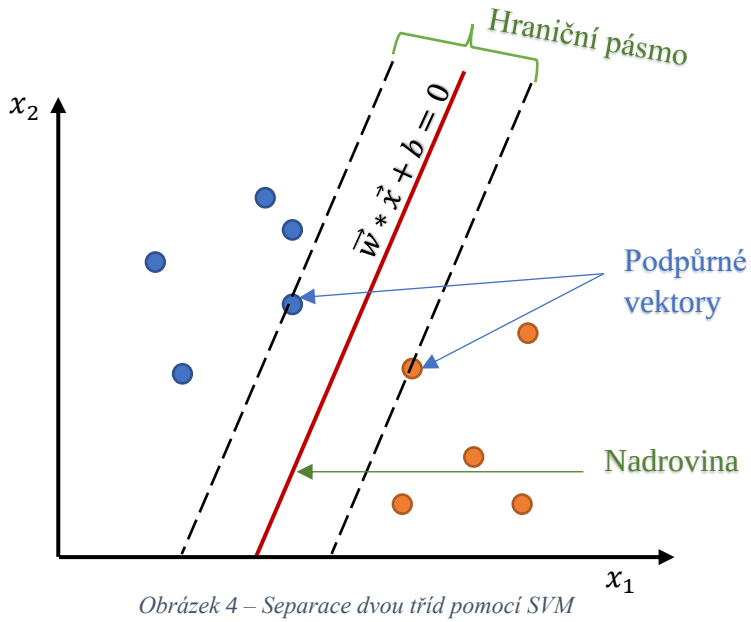


Obrázek 2 – Neseparovatelná data, obrázek převzat z [6]

Na obrázku 3 jsou znázorněna data dvou různých tříd, které jsou tzv. lineárně separovatelné. Pro data znázorněné v prostoru jsme schopni nalézt nadrovinu takovou, která rozdělí data na dvě části. Část pouze pro data z třídy první a část pro data pouze z třídy druhé. Pokud rozdělujeme data pouze do dvou tříd, hovoříme o tzv. binární klasifikaci [6].

Mějme lineárně separovatelnou množinu trénovacích dat $\{(\overrightarrow{x_i}, y_i), \dots, (\overrightarrow{x_N}, y_N)\}$, kde $\overrightarrow{x_i}$ je vektor atributů vzoru x_i a $y_i \in \{-1, +1\}$ je cílová třída vzoru. Cílem SVM je geometricky rozdělit trénovací data nalezením nadroviny.

$$< w, x > + b = 0 \tag{8}$$

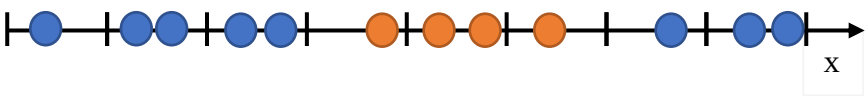


Obrázek 4 – Separace dvou tříd pomocí SVM

Kde w je normálový vektor nadroviny. Nalezení nadroviny $< w, x > + b = 0$ je proces optimalizační úlohy, jelikož nadrovin separující tyto dvě třídy může být nekonečně mnoho, hledáme takovou, která bude za pomocí podpůrných vektorů disponovat nejširším hraničním pásmem (maximum margin). Nové vektory $\vec{x_i}$ na vstupu pak klasifikujeme za pomocí následujících pravidel.

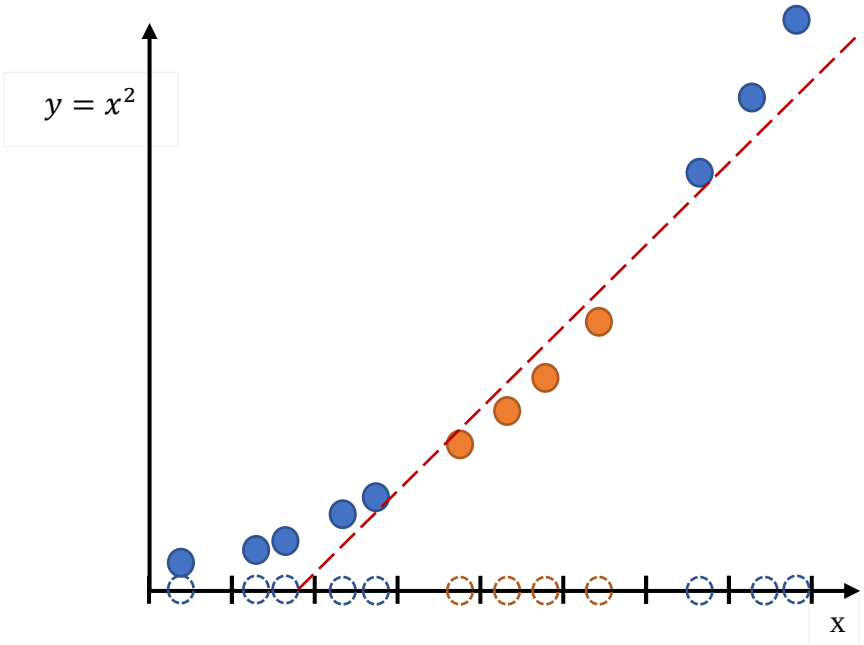
$$y_i \begin{cases} +1 \; pro < w, x_i > + b \geq 0 \\ -1 \; pro < w, x_i > + b < 0 \end{cases}$$

Otázkou zůstává, jak se klasifikátor zachová, pokud vstupní data nejsou lineárně separovatelná. Na začátku povídání o SVM jsem již uvedl, že tento druh klasifikátoru je velmi efektivní pro vícedimenzionální data. Svoje zásluhy na tom mají *kernelové funkce* (přezdívané také jako *jádrové funkce* [7]), díky kterým je možné přenést data z N dimenzionálního prostoru, ve kterém není možné data separovat, do prostoru $N + 1$, kde by separace mohla být možná. Kernelové funkce zajišťují transformaci dat do nové dimenze, aby mohla být data separována.



Obrázek 5 – Nelineárně separovatelný problém

Na obrázku 5 jsou znázorněna data v jedné dimenzi. V tomto případě, i když je shlukování zřejmé, nejsme schopni určit jednu hranici, která by tato data efektivně rozdělila na dvě skupiny patřičných tříd. V případě, že bychom se o to pokusili, klasifikátor by vykazoval špatné výsledky při určování tříd. Zkusme tedy tato data transformovat do nové dimenze pomocí $y = x^2$.



Obrázek 6 – Náčrt transformace dat do nové dimenze

Náčrt realizace $y = x^2$ z obrázku 6 ukazuje, jak původní jednodimenzionální lineárně neseparovatelný problém jsme po transformaci o dimenzi výš schopni separovat. Jinými slovy je SVM schopný najít nadrovinu takovou, která data efektivně rozdělí na dvě části a umožní relativně spolehlivou budoucí klasifikaci na netrénovaných datech.

Support Vector Machines ve skutečnosti nepřevádí všechna data do vyšších dimenzí, nýbrž pouze počítá vztahy mezi každým párem dat pomocí kernelových funkcí tak, jako kdyby se data nacházely v dimenzi vyšších, tyto vícedimenzionální vztahy umožní výpočet bez potřeby skutečného převádění dat a uleví tak výpočetnímu výkonu. Tato skutečnost se nazývá *The Kernel Trick* [8].

Nejběžněji používané kernelové funkce, převzato z [9]:

- **Lineární**: $K_{xx'} = K(x, x')$
- **Polynomická**: $K_{xx'} = K((\gamma\langle x, x'\rangle + r)^d)$, d je stupeň polynomu, r je konstanta
- **Radial Basis Funcion**: $\exp(-\gamma \parallel x - x' \parallel^2)$, pro $\gamma > 0$
- **Sigmoida**: $\tanh(\gamma\langle x, x'\rangle + r)$, kde r je konstanta

3.2 Předzpracování textu

Než se pustím do povídání o metodách word embeddingu, dovolím si stručné povídání o předzpracování textových dat. Předzpracování se obecně říká **Preprocessing** a je nedílnou součástí automatické kategorizace za použití technik strojového učení. Na počátku celého procesu stojí webová stránka. Jedná se o HTML dokument, ze kterého za použití technik web scrapingu vytáhneme veškerý text. Tento text se ovšem nachází v hrubé podobně. Této formě textu říkáme raw či také plain text.

1. **Lowering**
2. **Odstranění šumu** (noise removal)
3. **Tokenizace**

3.2.1 Lowering

První důležitou metodou text prerocessingu je **Lowering** (Lower Casing). Jedná se o převod veškerého textu na malé znaky, abychom v následné analýze nenaráželi na *Case Sensitivity* problém.

Raw/Plain text	Výsledek operace
Kočka kočkA KOČKA	kočka
Tomáš TOMÁŠ toMáš	tomáš

Tabulka 1 – Lowering textu

3.2.2 Odstranění šumu

Další dílčí metodou je tzv. **Noise Removal**, v překladu odstranění šumu. Šum se v textu vyskytuje v podobě nechtěných znaků, které defacto znemožňují efektivní textovou analýzu. Může se jednat o zapomenuté HTML značky, různé neabecední znaky jako jsou například pomlčky, čárky, podtržítko, vlnovka, číslice.

<a>kočka kočka_/ kočka2006	kočka

Tabulka 2 – Odstranění šumu z textu

3.2.3 Tokenizace

Výsledkem operace odstranění šumu již mohou být tzv. **Tokeny**, tedy jednotlivá slova namísto celých vět, množinu tokenů si lze snadno představit jako objekt List v Pythonu. Tokenizace je velmi důležitou operací pro klasifikační úlohy, jak již bylo zmíněno, vstupem do klasifikátorů jsou atributy vzoru (v případě procesu učení včetně cílové třídy), tyto atributy tvoří právě jednotlivé tokeny (slova, angl. features). Množina tokenů se nazývá **Korpus**.

Tokenizace vět může být velmi problematická, pokud věty obsahují slova spojena spojovníkem, proces totiž odděluje slova na základě určitých pravidel, může se jednat o mezery, tečky, čárky apod. [10]. Tento problém je znázorněn v druhém řádku následující tabulky.

Kočka sedí na okně.	Kočka, sedí, na, okně
Zapomeneme-li heslo k Wi-Fi	Zapomeneme, li, heslo, k, Wi, Fi

Tabulka 3 – Tokenizace textu, problém tokenizace některých výrazů

Nyní ještě zmíním metody, které sice nejsou nezbytně nutné pro účely klasifikace, ale ve většině případů velmi zlepší úspěšnost klasifikačních modelů. Ačkoliv se tyto metody již standartně objevují v angličtině, němčině a dalších jazycích světové úrovně, jsou tyto preprocessingové metody v českém jazyce velmi těžce dostupné. Nástroje pro tyto účely jsou většinou pouze proprietární.

3.2.4 Stematizace

Databáze Národní knihovny tento výraz vykládá jako „Automatické zkracování slov podle slovních kořenů.“ [11]. Anglická literatura hovoří o termínu Stemming. Proces stematizace provádí oddělení afixů (prefix, sufix, …) od lexémů (základní jednotkou slovní zásoby). Oddělení afixů vede k nalezení kmene slova [10]. Korpus složený z takovýchto tokenů následně vede k přesnější klasifikaci, protože modely neklasifikují téže slovo v mnoha tvarech jako samostatné tokeny, nýbrž jako jeden výraz.

3.2.5 Lemmatizace

Proces lemmatizace má za úkol převést slovo na tzv. *lemma,* tedy základní tvar tohoto slova. Lemma je reprezentativní podoba daného výrazu, v českém jazyce se používá také termín *heslové slovo* [12] a jedná se vždy o jednoslovnou jednotku. Lemmatizace

je součástí *desambiguace* (proces zjednoznačnění). Příkladem může být převedení slovesného tvaru *vytvoříme* na lemma *vytvořit* a tvary *lesům, lesy, lesích* na jednotné lemma *les* [13].

Původní výklad	Lemmatizace	Stematizace
textovým analýzám	textový, analýza	text, analýz
socio-ekonomického přínosu	socio, ekonomický, přínos	socio, ekonom, nos
chybovým kódem 403	chybový, kód, 403	chyb, kód, 403

Tabulka 4 – Příklad lemmatizace a stematizace. Převzato z: [10]

3.3 Číselná reprezentace textu

V této části se budu věnovat číselné reprezentaci webových stránek, respektive převodu textového obsahu na číselné hodnoty, vektory, se kterými si rozumí jednotlivé klasifikační modely. V předchozí části již bylo řečeno, že součástí množiny trénovacích a testovacích dat jsou jednotlivé atributy, které se předávají klasifikátorům, na základě těchto atributů je umožněno text zařadit do některé z cílových tříd. Převodu textu na číselné hodnoty se v angličtině říká *Word Embedding*. Existuje řada způsobů, jakými lze předkládat textové dokumenty na vstup klasifikačním algoritmům. V této části se budu věnovat těm momentálně nejrelevantnějším v kontextu úlohy automatické klasifikace webových stránek.

3.3.1 TF-IDF

Nejznámější a pro textovou klasifikaci také zřejmě nejpopulárnější metrikou je kombinace metod TF, které zastupuje slova Term Frequency (frekvence termu), a IDF, které stojí za Inverse Document Frequency (inverzní frekvence dokumentů). V této části se zaměřím na teoretický základ. Tvorbě vektoru příznaků pro webovou stránku a konkrétní implementaci TF-IDF, včetně slovníku, se budu věnovat v praktické části této práce.

Mějme dataset předzpracovaných dat $D = \{d_1, d_2, \dots, d_n\}$, kde jednotlivé dokumenty d_i jsou tvořeny vektory atributů (tokenů), $d_i = \left(a_1^{(i)}, a_2^{(i)}, \dots, a_m^{(i)}\right)$, kde a_j značí konkrétní token, $1 \leq j \leq m$, m je dimenze vektoru d_i. Složka TF se vypočítá následovně,

$$tf(a_k, d_i) = \frac{f\left(a_k^{(i)}\right)}{|d_i|} \tag{9}$$

kde čitatel $f\left(a_k^{(i)}\right)$ je frekvence právě počítaného termu (tokenu) $a_k^{(i)}$, a jmenovatel $|d_i|$ je počet všech termů v dokumentu d_i. Počítáme tedy frekventovanost tokenu na dané webové stránce.

Složka IDF se v našem kontextu počítá jako logaritmus podílu počtu všech webových stránek v datasetu ku počtu webových stránek s výskytem právě počítaného tokenu.

$$idf(a_k, D) = \log \frac{|D|}{|d \in D : a_k \in d|} \tag{10}$$

Tyto dvě varianty se běžně používají pro účely klasifikace textových dokumentu i jednotlivě. Nejpopulárnější a nejběžnější je ale právě již zmíněné TF-IDF, které se vypočte jako součin těchto dvou hodnot.

$$tfidf(a_k, d_i, D) = tf(a_k, d_i) * idf(a_k, D) \tag{11}$$

TF-IDF udává váhu jednotlivým termům a můžeme si ho tedy představit jako skóre, kterého dosáhl token, o který se právě zajímáme a běžně se používá pro vyhledávání v textu a klasifikační úlohy.

3.3.2 Word2Vec

Word2Vec jsou nejčastěji dopředné neuronové sítě, které umožňují nacházení souvislostí mezi jednotlivými vektory slov. Vynálezcem této metody je velmi významný český vědec Ing. Tomáš Mikolov, Ph.D., kterému byla v roce 2018 udělena Cena Neuron za významný vědecký objev.

Princip algoritmu vychází z myšlenky, že slova významově podobná jsou si v prostoru blíže, a naopak slova která podobnosti nevykazují, jsou od sebe více vzdálena. Jako příklad se často uvádí vektory slov o královi a královně.

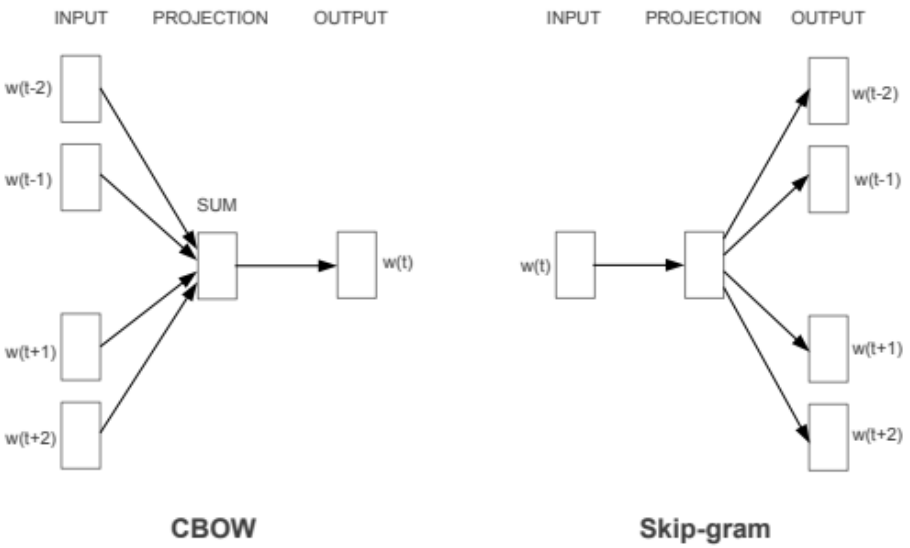
$$vec(\text{"queens"}) = vec(\text{"}kings\text{"}) - vec(\text{"}king\text{"}) + vec(\text{"}queen\text{"})$$

Jedná se tedy o odpověď na otázku *„King" se má ke „kings" jako se má „queen" k?* [14]

Dvě nejznámější implementace Word2Vec jsou:

- **CBOW** – Continuous Bag of Words je model který predikuje slova na základě slov předcházejících. Tato implementace se dnes hojně využívá například v klávesnicích chytrých telefonů, které nabízí slova na základě právě psané věty.
- **Skip-gram** – Principiálně se jedná se o přesný opak CBOW. Skip-gram model predikuje okolní slova na základě slova, které leží mezi nimi [14]. Technika vyžaduje daleko větší dataset.

Word2Vec díky své predikci slov našli využití také pro automatickou kompletaci kódu v programování, a to implementací přímo v patřičném prostředí (IDE), nebo v podobě různých pluginů [15].



Obrázek 7 – Architektury CBOW, Skip-gram. Převzato z článku: [21]

Technika Word2Vec bohužel není vhodná pro reprezentaci dokumentů (webových stránek), tato metoda je totiž schopna vektorizovat pouze jednotlivá slova v dokumentu. Vstupem klasifikátoru se předpokládá vektor příznaků, které reprezentují celý dokument a na základě těchto příznaků se realizuje funkce $f(x) = y$.

Jednotný rozměr vektoru příznaků jsme schopni snadno realizovat u techniky TF-IDF za pomocí slovníku. Poslední dobou se objevují techniky, kterými lze sumarizovat dokument i za pomocí Word2Vec, a to tak, že se jednotlivé vektory slov sloučí za pomocí funkce min, max, střední hodnota apod., těmito operacemi ovšem ztratíme spoustu informací, kvůli kterým bychom si Word2Vec vybrali. Této problematice se velmi

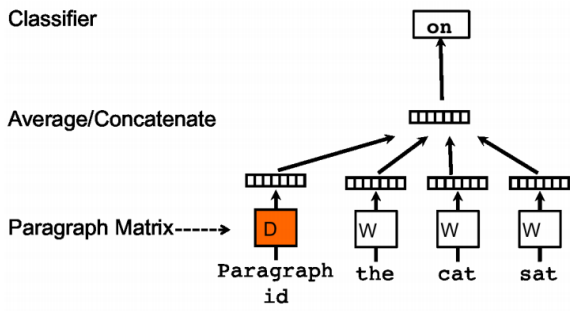
podrobně věnuje publikace Cedrica De Booma, Stevena Van Canneyta, Thomase Demeestera a Barta Dhoedta s názvem *Representation learning for very short texts using weighted word embedding aggregation* z roku 2016, kteří publikují průběh a výsledky po provedení jednotlivých operací [16].

3.3.3 Doc2Vec

Další techniku, kterou lze použít pro reprezentaci dokumentu, je tzv. Doc2Vec. Tuto metodu vynalezl stejný český vědec, který přišel s myšlenkou Word2Vec, ve spolupráci s vědcem Quoc V. Le. Vychází z jeho původního algoritmu Word2Vec, ovšem kvůli ztrátě kontextu z předešlé metody byl přidán další vektor, který se nazývá paragraph_id nebo také paragraph vector. Architektury modelu se dělí na:

- **PV-DM** (Distributed Memory Version of Paragraph Vector)

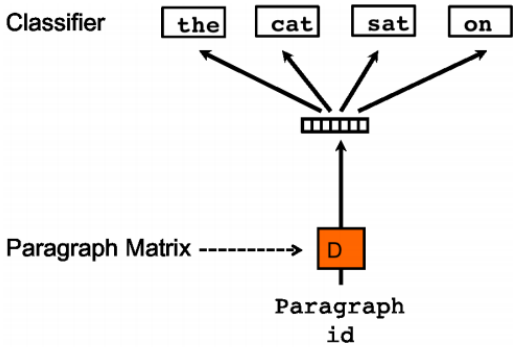
Tato architektura je velmi podobná CBOW architektuře z Word2Vec, přidán paragraph_id.



Obrázek 8 – Architektura PV-DM pro Doc2Vec. Převzato z: [22]

- **PV-DBOW** (Distributed Bag of Words Version of Paragraph Vector)

Druhý případ architektury Doc2Vec. Inspirován byl architekturou Skip-Gram, přidáno bylo paragraph_id.



Obrázek 9 – Architektura PV-DBOW pro Doc2Vec. Převzato z: [22]

3.4 Vyhodnocení úspěšnosti a testování klasifikátoru

Po procesu učení jednotlivých klasifikačních modelů je třeba modely patřičně otestovat, ať už kvůli vyhodnocení úspěšnosti, stanovení výsledků experimentů nebo také pro porovnání výsledků, které klasifikační modely vykazovaly při použití různých vstupních parametrů. Testování probíhá na testovací množině dat, ve které by se neměly opakovat vzory z množiny trénovací.

Většina metrik, které se pro účely vyhodnocování používají, vychází z tzv. **matice záměn** (confusion matrix), která zobrazuje správné odpovědi a odpovědi na výstupu klasifikátorů. V základním tvaru matice, tzn. pro binární klasifikaci, rozeznáváme čtyři druhy odpovědí – **true positive** (správně positivní), **true negative** (správně negativní), **false positive** (chybně pozitivní) a **false negative** (chybně negativní).

	Model Ano	Model Ne
Správně Ano	True Positive (TP)	False Negative (FN)
Správně Ne	False Positive (FP)	True Positive (TP)

Tabulka 5 – Matice záměn pro binární klasifikaci (Confusion matrix)

Z matice jsou zřejmé tři hlavní vlastnosti. Sloupce tvoří odpovědi klasifikátoru a řádky odpovídají správným odpovědím. Bezchybný klasifikátor má tedy nenulové hodnoty pouze na své hlavní diagonále. Pokud bychom sečetli všechny hodnoty matice, dostali bychom součet všech vzorů z testovací množiny.

Nejčastěji využívaná metrika pro vyhodnocení úspěšnosti klasifikátorů:

- **Celková přesnost** (Accuracy)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

- **Celková chyba** (Error)

$$Error = \frac{FP + FN}{TP + TN + FP + FN} \tag{13}$$

- **Přesnost** (Precision)

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

- **Úplnost** (Recall)

$$Recall = \frac{TP}{TP + FN} \tag{15}$$

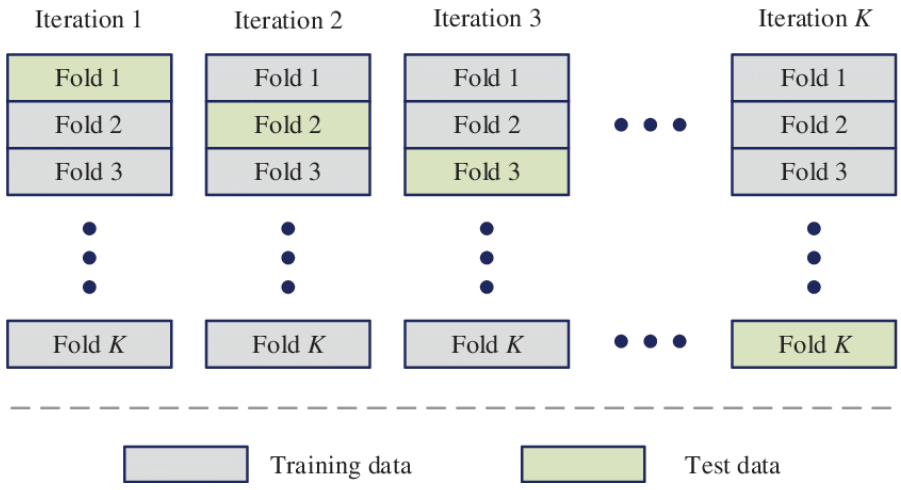
- **F-míra** (F-measure)

$$F\ measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{16}$$

K samotnému testování následně můžeme využít řadu metod, které s sebou nesou určité aspekty, např. princip testování či jaký poměr trénovacích a testovacích dat se použije.

3.4.1 Křížová validace

Křížová validace, anglicky **Cross Validation**, je jednou z nejznámějších a nejpoužívanějších metod testování. Princip metody je takový, že dostupná data rozdělíme na K částí. K-1 částí se použije pro trénování modelu a zbylá část se využije pro testování. Tento postup se K krát opakuje. Pro testování se pokaždé využije jiná část množiny. Výsledky dílčích operací se nakonec zprůměrují.



Obrázek 10 – Křížová validace K-Fold, převzato z článku: [23]

3.4.2 Leave One Out

Pro množiny dat s menším počtem vzorů se dá využít metoda Leave One Out, princip je pak dost podobný křížové validaci. Předpokládejme, že v máme množinu dat o N vzorech, N-1 vzorů se využije pro trénování modelu, zbylý vzor se následně využije pro testování. Proces se opakuje N krát a výsledky se průměrují. Leave One Out je výpočetně velmi náročnou metodou.

3.4.3 Náhodný výběr

Náhodný výběr je nejjednodušší a časově nejméně náročnou metodou pro testování. Data náhodně rozdělíme v určitém poměru na trénovací a testovací množinu. Výpočet s takhle rozdělenými daty proběhne jen jednou.

Nejčastěji používané rozdělení při náhodném výběru:

- Trénovací: 80 %, Testovací: 20 %
- Trénovací: 67 %, Testovací: 33 %
- Trénovací: 50 %, Testovací: 50 %

4 Praktická část

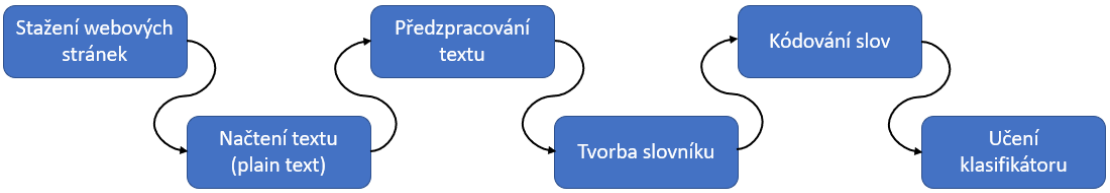
Tato část je věnována návrhu řešení a následné implementaci softwaru, který je schopen klasifikovat webové stránky do několika kategorií na základě textových informací, které se na webové stránce nacházejí.

4.1 Obecný návrh řešení

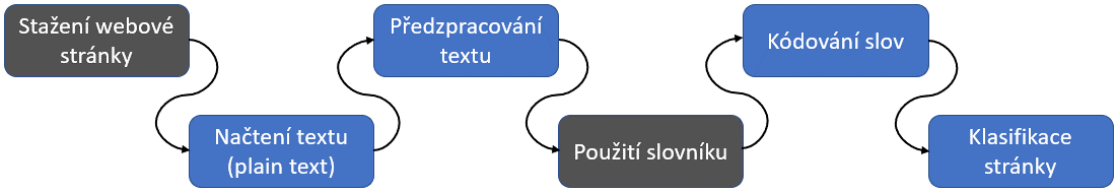
Než se pustím do samotné implementace, je nutné určit, jaké dílčí kroky budou potřeba k dosažení cíle a jaké typy úloh bude software řešit.

1. Vývoj celého řešení stojí v první řadě na **množině dat**, se kterou bude software následně pracovat. V tomto případě jsou to **webové stránky a kategorie**, do kterých patří. Správnost dat je při řešení klasifikačních úloh stěžejní a jejich dostupnost je nutnou podmínkou pro učení klasifikačních modelů.
2. Z množiny stránek je potřeba **stáhnout všechny webové stránky**. Za tímto účelem je důležité vybrat nástroj takový, který si umí poradit s dynamicky generovaným obsahem, popř. rámcovými weby.
3. Ze stažených stránek vytáhnout veškerý text. Text z webových stránek bude později podroben analýze. Stahování textu je předmětem **web scrapingu**.
4. Text se nachází v tzv. raw/plain podobě, aby byla klasifikace možná, resp. co nejvíce úspěšná, je potřeba tento raw text předzpracovat a **vytvořit množinu tokenů**.
5. Z tokenů vytvořit **slovník**, tedy **korpus**, celého klasifikačního algoritmu.
6. Vymyslet a naimplementovat způsob, kterými se jednotlivé **tokeny zakódují**. Výstupem této úlohy by měl být **vektor příznaků**.
7. **Vybrat metody strojového učení**, které po implementaci budou realizovat funkci $f(x) = y$, kde x bude vektor příznaků a y výsledná kategorie.

Samotný program by poté měl rozlišovat fázi učení či testování a fázi klasifikace jednotlivé stránky, která se objeví na vstupu programu s již naučenými klasifikátory.



Obrázek 11 – Workflow algoritmu při procesu učení



Obrázek 12 – Workflow algoritmu pro klasifikaci stránky

Na obrázku 11 a obrázku 12 si lze všimnout mírně pozměněných přístupů pro fázi učení a klasifikaci, kde v případě prvním dochází ke stahování celé množiny stránek a tvorbě slovníku. Ve druhém případě dochází ke stažení stránky, která se nachází na vstupu algoritmu a následně se použije slovník, který byl vytvořen ve fázi trénovací.

4.2 Implementace

Tato část je věnována konkrétnímu řešení včetně odpovědí na otázky z kapitoly 4.1

4.2.1 Sběr dat

Základním stavebním kamenem celého projektu jsou data, na kterých je prováděno učení a testování klasifikačních modelů. Z hlediska spolehlivosti klasifikace bylo zapotřebí sehnat dostatečné množství webových stránek a určit jejich cílovou kategorii. Tato dílčí část lze řešit v podstatě třemi způsoby.

1. Sehnat již vyhotovenou množinu stránek vč. cílových kategorií.
   - Pro české stránky velmi obtížné, momentálně neexistuje žádný volně dostupný zdroj a vyžaduje účast třetí strany, která takováto data dobrovolně poskytne.
2. Napsat script, jenž bude předmětem web scrapingu, pro stahování webových stránek z katalogů.
   - Katalogy webových stránek mohou být zastaralé. V lepším případě obsahují stránky, které již neexistují nebo jim vypršela platnost domény. V horším případě obsahují stránky, jejichž obsah neodpovídá předem deklarované cílové kategorii. Při řešení klasifikační úlohy nelze akceptovat nespolehlivá

data, data takového typu vedou k nesprávnému naučení modelu a tím pádem roste celková chyba klasifikace.

3. Nasbírat webové stránky ručně a ověřit platnost obsahu.
    - Ruční sběr je oproti předešlým metodám časově velmi náročný, nabízí ovšem možnost ověření platnosti a spolehlivosti obsahu stránek. Tyto vlastnosti jsou při řešení klasifikačních úloh klíčové.

Pro sběr dat jsem nejdříve zvolil možnost psaní scriptu pro stažení webových stránek z katalogů na adrese https://odkazy.seznam.cz/, tento postup by mi umožnil sehnat velké množství stránek do svého datasetu. Později jsem došel ke zjištění, že katalog je velmi zastaralý. Velké množství odkazů již vůbec nefunguje a v mnoha případech stránky neobsahovaly předem deklarovaný obsah, nýbrž obsah z úplně jiné kategorie. **Zvolil jsem tedy cestu ručního sběru.** Jelikož klasifikace probíhá na základě analýzy textu, stránky musely rovněž splňovat podmínku přítomnosti dostatečného množství smysluplných textových informací, ze kterých lze rozeznat, o jakém tématu stránka pojednává. Z toho rovněž vyplývá, že stránky složené pouze z obrázků jsem do datasetu nezařadil vůbec.

Samotný sběr stránek probíhal tak, že jsem vybral kategorii, která obsahovala dostatek webů, jednotlivě jsem procházel všechny odkazy, které náležely této kategorii (či podkategorii) a webové stránky, které byly relevantní a vyhovující pro moji úlohu, jsem uložil do oblíbených položek s názvem cílové kategorie. Po nasbírání všech odkazů jsem pouze již vyexportoval oblíbené položky do HTML souborů tak, že jeden HTML soubor se rovnal jedné kategorii se všemi weby, které této kategorii náleží. Tato cesta se mi zdála v kontextu ručního sběru nejrychlejší.

4.2.2 Cílové kategorie

Bylo předem jasné, že kvůli časové náročnosti nebudu schopen nasbírat velké množství kategorií, které by obsahovaly dostatek odkazů. Vyhledal jsem proto konkrétní kategorie, pro které, v katalogu webových stránek od firmy Seznam.cz, a.s., byla předem deklarována existence alespoň 400 odkazů. Toto snažení mě přivedlo k následujícím kategoriím:

- **Psi**
- **Kočky**
- **Gastronomie**
- **Fotbal**
- Ostatní (přidána později)

Přesně tyto kategorie dokázaly splnit podmínku, aby se jednalo o konkrétní téma, ve kterém bude obsaženo nejméně 400 odkazů. Podotýkám, že u kategorie kočky jsem musel přimhouřit oči, protože bylo původně deklarováno o něco méně vzorků než 400. Po ručním sběru a kontrole obsahu mi z každé kategorie zbylo alespoň 240 vzorků pro každou kategorii.

S těmito čtyřmi kategoriemi, které jsou zvýrazněny tučným písmem, později probíhala velká část experimentu, později se ale vyskytla otázka: „Co se stane, pokud ke klasifikaci přiložím stránku z úplně jiné kategorie?“. Klasifikátor stránku zařadí do některé z existujících kategorií, které se naučil na trénovacích datech. Proto jsem později nasbíral další kategorii, která je složena z náhodných webových stránek, které nespadají ani do jedné ze čtyř předešlých. Zbývala pouze otázka, jak tato obecná kategorie, vložená do čtyř konkrétních, ovlivní úspěšnost klasifikace.

	Kategorie	Původní počet stránek (deklarováno katalogem)	Počet stránek zařazených do datasetu
1	**Psi**	2709	401
2	**Kočky**	376	248
3	**Gastronomie**	551	267
4	**Fotbal**	626	240
5	**Ostatní**	-	276

Tabulka 6 – Kategorie pro klasifikaci

4.2.3 Programové vybavení

Jako programovací jazyk jsem pro tvorbu softwaru zvolil Python, a to především kvůli rychlosti, flexibilitě a dostupnosti knihoven a nástrojů, které se pro machine learning a text mining běžně používají a které znatelně ulehčí vývoj tohoto softwaru.

Programové vybavení, externí knihovny, užití:

- **Python** 3.9.1 – programovací jazyk
- **beautifulsoup4** v.4.9.3 – web scraping
- **numpy** 1.20.1 – matematické výpočty
- **pandas** 1.2.2 – efektivní práce s textem
- **scikit-learn** 0.24.1 (**sklearn**) – machine learning, text mining
- **selenium** 3.141.0 – stahování webových stránek
- **joblib** 1.0.1 – ukládání a načítání naučených modelů, serializace objektů

4.2.4 Struktura programu

Tento software využívá klasifikační modely **SVM**, **Naive Bayes** a **K-Nejbližší sousedů**. Pro vektorizaci textu (tvorbu příznaků) je využívána metodika **TF-IDF**. Program je konstruován celkem z osmi tříd a jednoho spouštěcího scriptu. Každá z níže uvedených tříd má v programu specifickou funkci a obstarává pouze typ úlohy, která mu náleží. Kompletní zdrojové kódy včetně dokumentace a návodem na spuštění jsou součástí přílohy práce.

Třídy:

- **`ScrapeHandler`** – Obstarává veškerou práci s HTML dokumenty, včetně stahování textů z webových stránek.
- **`DatasetManager`** – Připravuje a předzpracovává stažená data pro následnou klasifikaci, stará se o tvorbu korpusu nad textem z webových stránek.
- **`DatasetVectorTfIdf`** – Třída zahrnující metodiku TF-IDF. Slouží ke tvorbě TF-IDF slovníku, rozdělení dat na trénovací a testovací množinu, vektorizaci textu (tvorba příznaků) pro vstup do klasifikátorů.
- **`Classifier(ABC)`** – Abstraktní bázová třída zahrnující základní proměnné a metody pro konkrétní klasifikátory. Společná část pro všechny klasifikační modely.

- `NaiveBayes(Classifier)` – Třída dědící z abstraktní třídy Classifier, inicializuje a specifikuje klasifikační model Naive Bayes.
- `SVM(Classifier)` – Inicializuje a specifikuje klasifikační model Support Vector Machines.
- `KNearestNeighbors(Classifier)` – Specifikuje a inicializuje model K-Nejbližší sousedů.
- **`ProgramControl`** – Třída obsahuje metody pro ovládání celého programu, ty jsou volány spouštěcím scriptem.
- **`main.py`** – Úvodní spouštěcí script, který obsahuje konzolové uživatelské rozhraní. Pro plnění požadavků uživatele vytváří objekt typu `ProgramControl`.

4.2.5 ScrapeHandler

Je třída, která obstarává veškerou práci s webovými stránkami a lokálně uloženými HTML soubory. Množina webových stránek byla sbírána vkládáním webů do oblíbených položek, které se následně vyexportovaly z prohlížeče a byly uloženy na lokální médium ve tvaru: *kat_název1.html, ..., kat_názevN.html.* Z tohoto důvodu je potřeba z takto vyexportovaných položek vytáhnout všechny odkazy. Tato činnost podléhá disciplíně zvané web scraping, pro kterou používám knihovnu **BeautifulSoup4** (dále jen bs4). Tato knihovna také stahuje všechny textové informace z webových stránek.

Třída `ScrapeHandler` původně využívala knihovnu *requests*, která obstarávala stahování html souborů z webových stránek. Tato knihovna je pro své účely velmi rychlá, ale nedokáže si poradit s webovými stránkami, které jsou psané rámcově. Z toho důvodu jsem knihovnu nahradil nástrojem **Selenium**. Tento nástroj využívá svůj ovladač (driver), kterým poskytuje rozhraní mezi Python scriptem a zvoleným prohlížečem. Původně byl tento nástroj vyvinut pro testování webových aplikací, avšak kvůli možnosti přistupování k dynamicky generovaným webům, jsem tento nástroj použil pro stahování dynamických a rámcových webových stránek. Pomocí parametrů lze ovladači říct, jaký způsobem má k webovým stránkám přistupovat.

Inicializace Selenium Webdriveru:

```
options = webdriver.ChromeOptions()

options.add_argument('--ignore-certificate-errors')
options.add_argument('--incognito')
options.add_argument('--headless')

des_capabilities = DesiredCapabilities.CHROME
des_capabilities["pageLoadStrategy"] = "eager"

driver_instance = webdriver.Chrome(
                    driver_path,options=options,
                    desired_capabilities=des_capabilities
)
driver_instance.set_page_load_timeout(driver_timeout)
```

Zdrojový kód 1 – Inicializace Selenium Webdriver

Ve zdrojovém kódu 1 můžeme vidět řadu argumentů. Webdriveru v podstatě říkáme, aby pro přístup k webům použil prohlížeč Google Chrome, dále aby ignoroval chybné certifikáty, stránky se mají otevírat v soukromém režimu, při navázání spojení neotevírat prohlížeč, načíst pouze text, obrázky nikoliv.

Takto inicializovaný webdriver lze následně použít pro získání html dokumentu konkrétní webové stránky, a to voláním metody **`driver.get(url)`**. Po zisku tohoto html dokumentu se nejdříve testuje, zdali je web psaný rámcově, pokud ne, bs4 rovnou stáhne text. Pokud by se jednalo o rámcovou webovou stránku, dá se předpokládat, že nejužitečnější informace se nachází na prvních třech rámcích. Selenium tedy získá textové informace z prvních třech rámců, a to pomocí přepínání kontextu webové stránky.

```
framesList = soup.findAll("frame")
if (len(framesList) < 1):
      self.__text_from_website = soup.getText()
else:
      for i in range(0, 3):
      driver.switch_to.default_content()
      driver.switch_to.frame(i)
      soup = BeautifulSoup(driver.page_source, "html.parser")
      self.__text_from_website += f"\n{soup.getText()}"
```

Zdrojový kód 2 – Úryvek metody pro získání textu z webové stránky

Stažený text je následně ve své raw/plain podobě předán třídě `DatasetManager` k dalšímu zpracování.

4.2.6 DatasetManager

Tato třída obstarává veškerou práci s textem, tzn. předzpracování a tvorbu hlavního korpusu. Třída je v celém programu nejobsáhlejší, z tohoto důvodu popíšu jen obecný princip této třídy, zdrojový kód programu je dostupný ve formě přílohy.

Jak jsem již zmínil, tato třída obstarává předzpracování textových dat. V tomto programu využívám následující metody:

- **Lowering** – změna na malé znaky napříč textem
- **Odstranění šumu** – odstranění nechtěných či překážejících znaků (vč. číslic)
- **Tokenizace** – vytvoření kolekce tokenů (slov)
- **Odstranění stopwords** – odstranění slov s nulovou vypovídající hodnotou

Množina stopwords byla převzata z [17]. Některé výrazy byly pro účel mé úlohy přidány, jedná se o výrazy – *dát, www, přihlásit, přihlášení, přihlá, teď*. Naopak tato slova byla ze seznamu odebrána – *chuť, clanek, clanku, clanky, jím, téma*.

Klasifikace probíhá na základě N nejfrekventovanějších slov. Abychom využili potenciál metodiky TF-IDF, je třeba všechna tato nejfrekventovanější slova uložit do korpusu tolikrát, kolikrát se na webové stránce vyskytly. Hlavní úkol této třídy je pak následující.

1. Pokud jsou data uložena lokálně, pokračuj bodem 4.
2. Ze třídy ScrapeHandler si vyžádej text z každé stránky + kategorii, do které stránka patří.
3. Tato data ulož lokálně do souboru progressedData.txt.
4. Zjisti parametr N.
5. Načti obsah souboru progressedData.txt.
6. Proveď předzpracování textových dat.
7. Vytvoř histogram tokenů (slov)
8. N nejfrekventovanějších tokenů ulož do korpusu tolikrát, kolikrát se na stránce vyskytly.
9. Ulož korpus do souboru corpus_top_words.csv.

Pokud by se parametr $N = 2$, výsledný korpus by měl následující formát:

```
category     ;     text
0     ;     formát formát formát formát textu textu
0     ;     řádek řádek řádek druhý
1     ;     druhá druhá druhá kategorie kategorie
…     ;     …
```

Zdrojový kód 3 – Příklad výsledného korpusu pro K=2

Takto zkonstruovaný korpus je následně předán třídě `DatasetTfIdf` pro tvorbu TF-IDF slovníku, pomocí kterého jsou tvořeny vektory příznaků.

```
def getInputTopWords(self, top_words):
      len_top_words = len(top_words)
      text = "category" + self.SEPARATOR + "text\n"
      text+= f"-1{self.SEPARATOR}"
      for i in range(0,len_top_words):
            text +=(top_words[i][0] + " ") * top_words[i][1]
      text_for_pandas = io.StringIO(text)
      text_dataframe = pd.read_csv(
            text_for_pandas,sep=self.SEPARATOR,
            engine="python", encoding="utf-8"
      )
      return text_dataframe
```

Zdrojový kód 4 – Získání cílového formátování v rámci predikce

V případě, že by si některá z klasifikačních tříd vyžádala konstrukci textu jedné webové stránky v rámci predikčního procesu, třídní metoda ze zdrojového kódu 4 vytvoří DataFrame, jenž je kolekcí knihovny Pandas, a zkonstruuje formát, který koresponduje s formátem souboru *corpus_top_words.csv*. Výstup následně putuje k zakódování, o který se stará třída `DatasetTfIdf`.

4.2.7 DatasetTfIdf

Třída obsahující metody pro tvorbu výsledného datasetu, který je přijímán samotnými klasifikátory. Obstarává rozdělení dat do trénovací a testovací množiny, tvorbu TF-IDF slovníku a vektorizaci textu (tvorbu příznaků) vůči TF-IDF slovníku.

Tato třída využívá knihovnu **sklearn** z rodiny **scikit-learn**. Třídy pro práci s TF-IDF jsou implementovány v souboru **`sklearn.feature_extraction.text`**. Počítání složky TF (*term frequecny*) má na starost třída **`CountVectorizer`**, která počítá frekvenci termu v rámci celé množiny dat (postupně přes každý vzor). Každý **unikátní token** dostane

své identifikační číslo (index), následně se vytvoří $M * N$ matice, kde řádky tvoří indexy unikátních termů, řádky pak zastupují vzory z trénovací/testovací množiny. Počet unikátních tokenů, kterým říkáme příznaky, určuje parametr max_features. Pokud bychom tedy měli korpus s 15000 unikátními slovy a určili max_features=10000, velikost TF-IDF slovníku by byla 10000 unikátních slov (tokenů/příznaků).

token/vzor	token 1	token 2	…	token M
vzor 1	3	0	1	0
vzor 2	0	2	3	1
…	0	0	0	1
vzor N	3	1	0	0

Tabulka 7 – Příklad TF matice vytvořené třídou CountVectorizer

Hodnoty matice z tabulky 7 poté reprezentují frekvenci výskytu daného tokenu(termu).

Počítání složky IDF má na starost třída `TfIdfTransformer`, ta přebírá vyhotovenou matici od třídy `CountVectorizer` a hodnoty v ní přetransformuje z formy TF na formu TF-IDF. Následující výpočty jsou převzaty z [18]:

$$idf(t) = log\frac{1+n}{1+df(t)} + 1 \tag{17}$$

kde n je celkový počet webových stránek, $df(n)$ je počet webových stránek s výskytem tokenu t. Rovnice se oproti učebnicovému příkladu mírně liší. Výsledné $tfidf$ vektory jsou následně normalizovány Eukleidovskou formou:

$$v_{norm} = \frac{v}{\|v\|} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}} \tag{18}$$

Původní hodnoty matice z tabulky 7 jsou tedy nahrazeny hodnotami TF-IDF z rozsahu $< 0; 1 >$, matice příznaků je následně přivedena ke klasifikačním modelům k procesu učení/testování.

```
# Argumenty jsou přítomny jako parametry konstruktoru
self.corpus = corpus
self.train_x, self.test_x, self.train_y, self.test_y =
       model_selection.train_test_split(
              self.corpus["text"], self.corpus["category"],
              test_size=test_size
)
self.tf_idf_vector = TfidfVectorizer(max_features=max_features)
# Nauč se slovník unikátních tokenů (features)
self.tf_idf_vector.fit(self.train_x)
# Vytvoř matice příznaků pro trénovací a testovací množinu
self.train_x_idf = self.tf_idf_vector.transform(self.train_x)
self.test_x_idf = self.tf_idf_vector.transform(self.test_x)
```

Zdrojový kód 5 – Úryvek konstruktoru třídy DatasetTfIdf

Zdrojový kód 5 ukazuje další třídu z knihovny sklearn. Jedná se o třídu **`TfIdfVectorizer`**, tato třída v sobě zahrnuje `CountVectorizer` a `TfIdfTransformer`, ulehčí tak práci programátorovi.

```
def vectorizeInput(self, text: DataFrame):
       return self.tf_idf_vector.transform(text["text"])
```

Zdrojový kód 6 – Použití slovníku k zakódování textu pro predikci

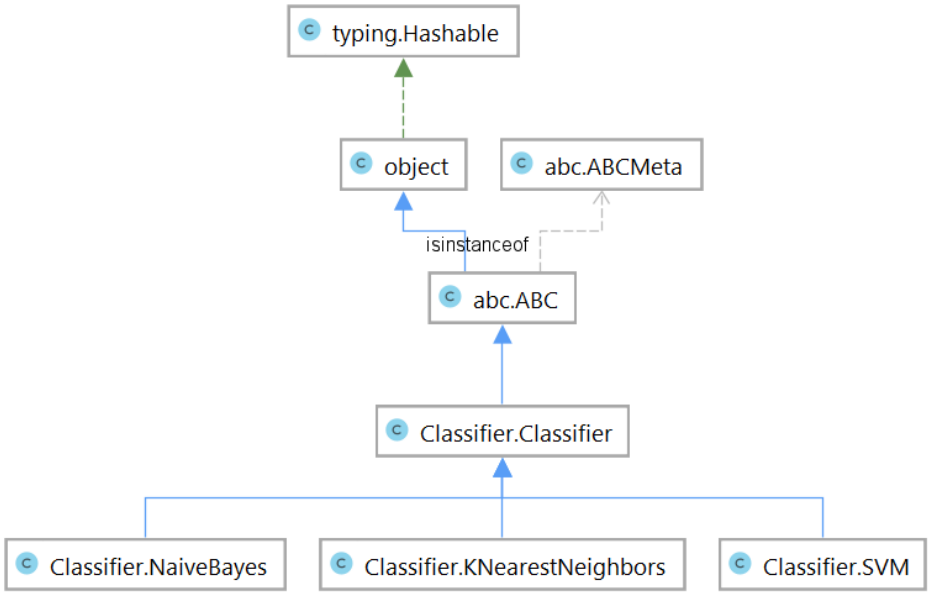
Ve zdrojovém kódu 6 si můžeme všimnout další metody z třídy `DatasetTfIdf`. Jedná se o metodu, která v procesu predikce webové stránky navazuje na zdrojový kód 4. Na vstupu přijímá objekt typu DateFrame, ve kterém se nachází text webové stránky. Formátování tohoto textu již koresponduje s formátováním hlavního korpusu díky přechozímu kroku. Tento text se zakóduje pomocí naučeného TF-IDF slovníku (proces ze zdrojového kódu 5) a vznikne tak vektor příznaků, který je přijímán klasifikačními modely.

4.2.8 Classifier

Třída Classifier je abstraktní bázovou, není tedy možné vytvořit instanci této třídy. Jedná se o společnou část implementace ostatních klasifikačních tříd NaiveBayes, SVM a KNearestNeighbors.

Python v základu neobsahuje anotace pro vytváření abstraktních tříd, za tímto účelem později vznikla třída ABC (*abstract base class*), ze které nelze vytvořit instanci, ale stále umožňuje navazující dědičnost. V případě, že u právě konstruované třídy přímo uvedeme dědění z ABC, třída zdědí vlastnost restrikce instanciování objektů. Tento princip tedy plně nahrazuje původní absenci abstraktních tříd a umožní předepsání společné části [19].

Struktura tříd z rodiny Classifier má následující podobu:



Obrázek 13 – UML Diagram tříd z rodiny Classifier

Třída Classifier obsahuje metody pro trénování či testování modelů a volání predikce pro webovou stránku, implementace je realizována pomocí knihovny **sklearn**. Třída rovněž obsahuje vybavení pro ukládání a načítání natrénovaných modelů, a to pomocí knihovny **joblib**, která umožňuje serializaci hotových objektů.

Třídní proměnná `self._model` reprezentuje konkrétní klasifikátory, tato proměnná je inicializována až samotnými potomky.

```
# Úryvek třídní metody Classifier::train_model(…)
# Natrénuj model. Dataset je objekt typu DatasetTfIdf
self._model.fit(dataset.train_x_idf, dataset.train_y)
# Otestuj model na testovacích datech
predict_of_model = self._model.predict(dataset.test_x_idf)
# Spočítej celkovou přesnost modelu [%]
result = accuracy_score(predict_of_model, dataset.test_y) * 100
```

Zdrojový kód 7 – Úryvek metody train_model(...) třídy Classifier

Metoda, jejíž úryvek je vyobrazen ve zdrojovém kódu 7, je volána svými potomky, kteří nejprve inicializují proměnnou `self._model`, následně volají rodičovskou metodu a proběhne proces učení klasifikátoru.

Hlavní metodou pro testování klasifikátorů je v tomto softwaru křížová validace, která nejprve rozdělí data na trénovací a testovací množinu, následně naučí model a poté vypočítá celkovou přesnost klasifikace, tento proces se opakuje podle zadaného parametru *cv*.

```
def cross_validate_score(self, max_features = 5000, cv=10):
      self.set_dataset(max_features,test_size=0.001)
      dataset =  self._dataset
      X = dataset.train_x_idf
      y = dataset.train_y
      # Rozděl dataset a proveď cross-validate knihovnou sklearn
      scores = cross_val_score(self._model, X, y, cv=cv)
      print(scores)
      print("%0.2f accuracy with a standard deviation of %0.2f" %
            (scores.mean(), scores.std()))
```

Zdrojový kód 8 – Křížová validace, kód částečně přejat z: [24]

Protože je třeba zajistit, aby došlo k transformaci dat, kterou má na starost objekt typu `TfIdfVectorizer` společně s metodou `train_test_split(…)` z knihovny sklearn, rozhodl jsem se obětovat nepatrný zlomek korpusu, konkrétně 0,01 %, který se křížové validace nezúčastní, tento počin je vyobrazen na 2. řádku zdrojového kódu 8.

4.2.9 ProgramControl, main.py

ProgramControl je třída, která se stará o funkčnost celého programu. Poskytuje dílčí metody, díky kterým je možné celý program inicializovat a následně řídit. Jedná se o prostředníka mezi klasifikačními třídami a uživatelským rozhraním, které se nachází v hlavním spouštěcím scriptu main.py. Uživatel zadá svůj požadavek prostřednictvím spouštěcího scriptu, script main.py se s touto žádostí obrátí na objekt třídy ProgramControl, který následně požadavek realizuje.

```
…\Složka s programem> python main.py -clf=svm https://www.jcu.cz
```

Zdrojový kód 9 – Požadavek na klasifikaci webové stránky pomocí SVM

```
…
elif opt_given == "-clf":
       if (param_given == Classifier.SVM):
              clf = Classifier.SVM
       elif (param_given == Classifier.KNN):
              clf = Classifier.KNN
…
       try:   #pc je instancí třídy ProgramControl, args list argumentů
              pc.predict_webpage(args[0], clf)
```

Zdrojový kód 10 – Úryvek spouštěcího scriptu při požadavku na klasifikaci

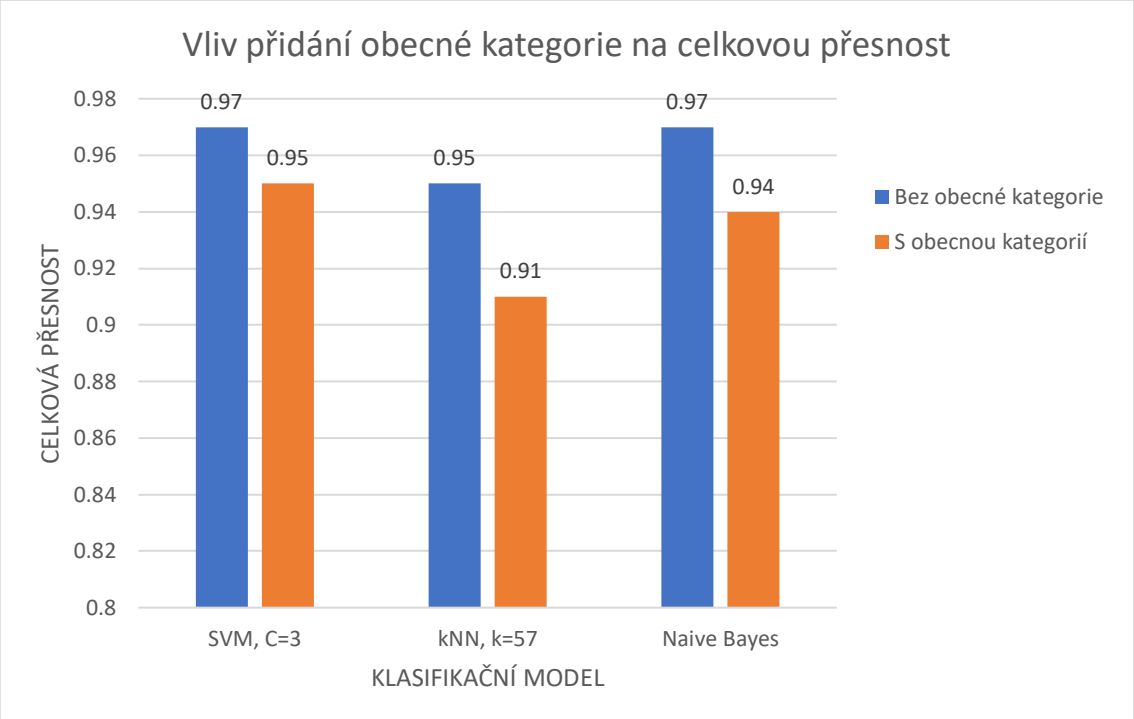
Spouštěcí script provede extrakci možností a parametrů zadaných na vstupu. Zjistí, že se jedná o příkaz ke klasifikaci webové stránky pomocí modelu SVM, požádá tedy objekt třídy ProgramControl o provedení této akce. Tento princip funguje analogicky u všech ostatních požadavků, které uživatel zadává.

4.3 Testování a výsledky

Po navržení a implementaci systému bylo provedeno testování celkové přesnosti modelu pro několik aspektů. Hlavní metrikou použitou pro testování byla **celková přesnost modelu** (accuracy) a jako techniku testování jsem zvolil **desetinásobnou křížovou validaci.**

4.3.1 Přidání obecné kategorie

Jak jsem již avizoval v podkapitole 4.2.2, původní počet kategorií byl o jedna menší, v množině webových stránek o *psech*, *kočkách*, *gastronomii* a *fotbalu* chyběla kategorie *ostatní*, která byla přidána později pro případ, že by se na vstupu objevila stránka, která nespadá ani do jedné ze čtyř uvedených kategorií. Přidání velmi obecné kategorie do čtyř konkrétních by mohlo mít za následek rapidní nárust celkové chyby klasifikace, z tohoto důvodu jsem na původních datech (4 kategorie) provedl měření celkové přesnosti pro všechny implementované modely a následně změřil celkovou přesnost modelů po přidání obecné kategorie ostatní.



Graf 1 – Vliv přidání obecné kategorie do čtyř konkrétních

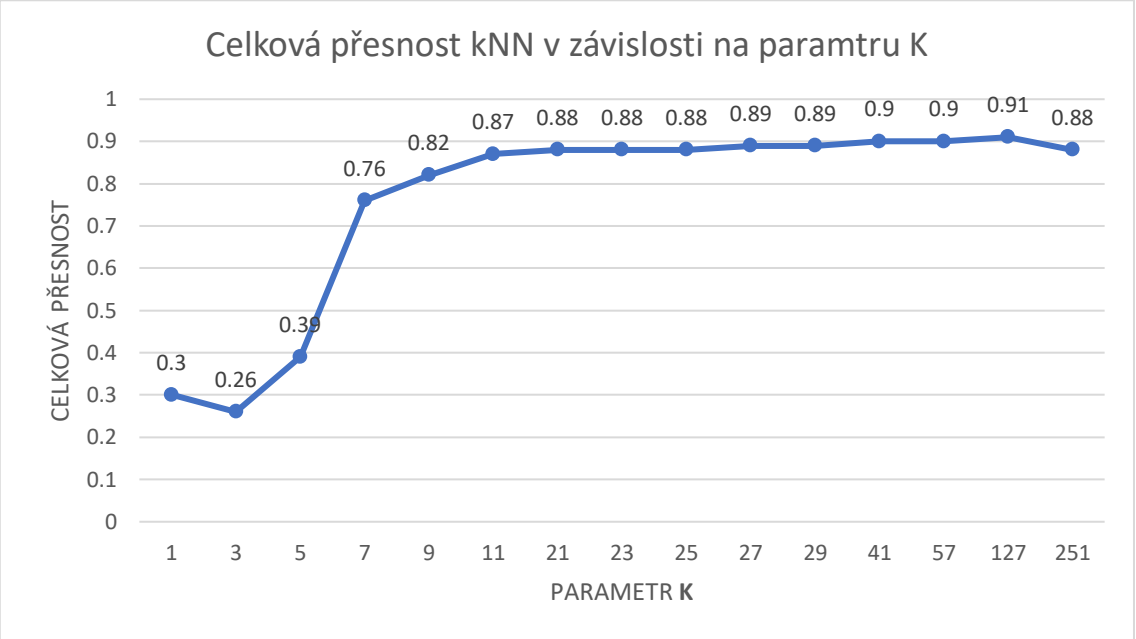
Při testování byly u modelů využity parametry, pro které modely vykazovaly nejvyšší úspěšnost při relativně malém nárustu výpočetní náročnosti. Z grafu 1 je zřejmé, že přidání obecné kategorie má vliv na procentuální úspěšnost modelu. Nutno podotknout,

že nárůst chyby není ani zdaleka tak razantní, jak by se předem dalo očekávat a že pokles celkové přesnosti je pravděpodobně zapříčiněn pouze přidáním kategorie, přičemž obecnost této kategorie nemusí mít až takový vliv.

4.3.2 K-Nejbližších sousedů

Předmětem tohoto testování bylo nalezení optimální hodnoty parametru K, které u algoritmu kNN udává, kolik sousedních vektorů z trénovací množiny určuje vítěze, tedy cílovou třídu nově klasifikovaných dat. Testovány byly pouze liché hodnoty.

Všechna následující měření byla provedena za přítomnosti obecné kategorie *ostatní*, tato kategorie je aktuálně součástí množiny stránek a experimentálního softwaru dodaného formou přílohy této práce.



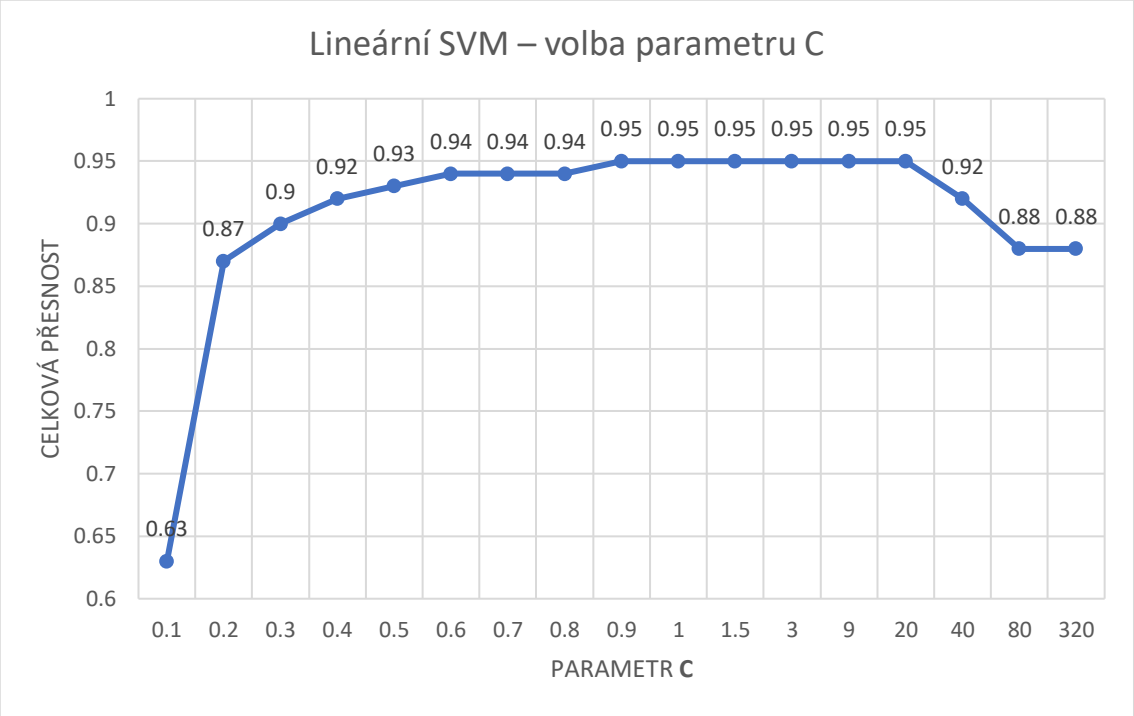
Graf 2 – K-Nejbližších sousedů, testování parametru K

Nalezení optimální hodnoty K je klíčové pro celý klasifikační algoritmus. Vyšší hodnoty v mnoha případech vedou k přesnější klasifikaci nových dat, nelze ale tuto hodnotu zvolit libovolně vysokou, jelikož s nárustem hodnoty K narůstá rovněž výpočetní náročnost. Z grafu 2 je zřejmé, že nejkritičtější rozdíly přesnosti klasifikace se dějí u nižších hodnot K, kdežto vysoké hodnoty mají relativně stabilní průběh.

Pro všechny ostatní měření bylo zvoleno $K = 57$, protože vykazovalo nejstabilnějšími výsledky při relativně nízké výpočetní náročnosti, a to se standardní odchylkou 0,01. $K = 57$ tvoří v experimentálním softwaru defaultní hodnotu.

4.3.3 Support Vector Machines

SVM implementované knihovnou sklearn disponuje širokou škálou nastavitelných parametrů. V této práci je pro klasifikaci využíváno SVM s lineární kernelovou funkcí. Pro tento typ je kritický především parametr C, který udává regulaci hraničního pásma tvořeného podpůrnými vektory. U implementace SVM knihovnou sklearn platí, že nižší parametr C znamená nižší regulaci hraničního pásma a připouští tak určitou chybu klasifikace. Nižší hodnoty C jsou využívány pro data, která částečně naráží na problém lineární separace [20].

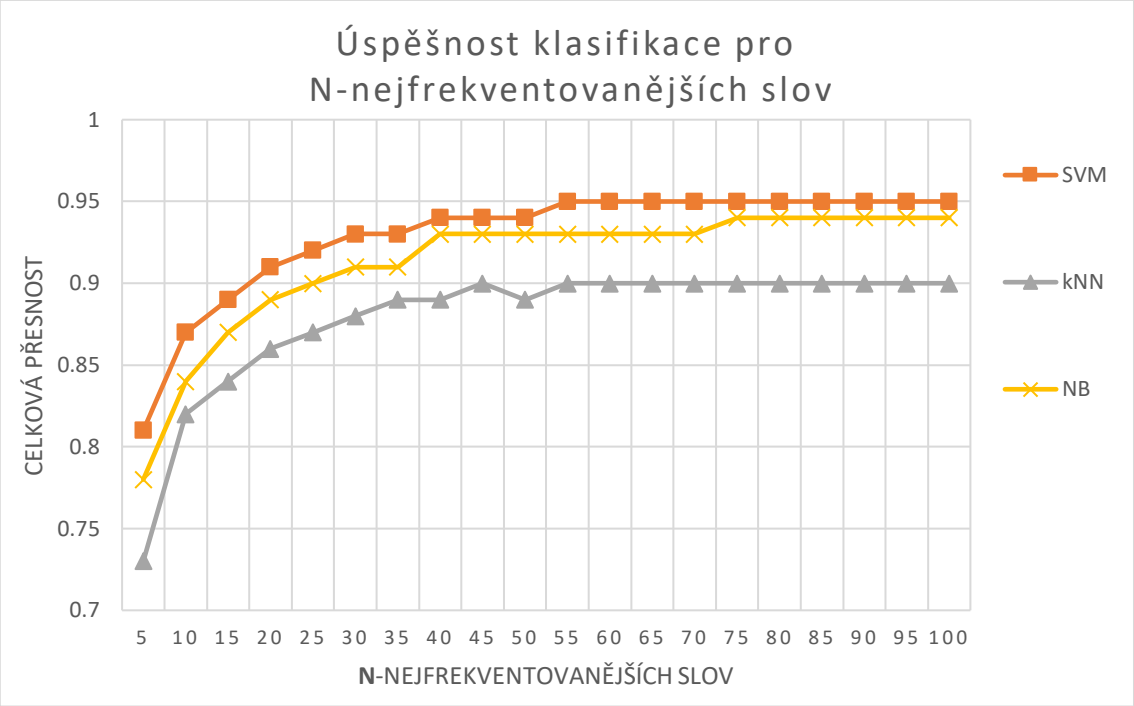


Graf 3 – Support Vector Machines, testování parametru C

Defaultní hodnota pro experimentální software je nastavena na C=3, znovu z důvodů nejstabilnějších výsledků při testování metodou křížové validace.

4.3.4 N-Nejfrekventovanějších slov

Celý klasifikační algoritmus stojí na extrakci N-nejfrekventovanějších slov z webových stránek v procesu předzpracování dat, nabízí se tedy velmi důležitá otázka – jaký vliv má parametr N na celkovou přesnost všech klasifikačních modelů?



Graf 4 – Porovnání modelů při N-nejfrekventovanějších slovech

Graf 4 ukazuje celkovou přesnost všech klasifikátorů. Z grafu je zřejmé, že největší vypovídající hodnotu má 10 nejfrekventovanějších slov. U této hodnoty N je nejvyšší nárust celkové přesnosti. Naopak u $N = 55$ se nárůst celkové přesnosti v podstatě zastaví a následně stabilizuje.

Hodnota N má přímý dopad na počet (ne)nulových hodnot v matici příznaků, tím pádem s vyššími hodnotami N stoupá výpočetní náročnost. V softwaru je defaultně nastavena na $N = 80$

5 Závěr

Tato práce se zabývala klasifikací webových stránek a byla členěna na část teoretickou a část praktickou.

V rámci teoretické části byla provedena literární rešerše v oblasti strojového učení, kde byly podrobně popsány použité klasifikační modely, a to Naivní Bayesuv klasifikátor, K-Nejbližších sousedů a Support Vector Machines. V teoretické části byla rovněž popsána problematika text miningu v rámci českého jazyka. Následně se teoretická část zabývala číselnou reprezentací textu, konkrétně nejpopulárnější metodou TF-IDF. V rámci nových přístupů ke klasifikaci webových stránek byly popsány i další metody číselné reprezentace, a to Word2Vec či Doc2Vec, které by při správném využití dokázaly zachytávat podrobnosti slov a detekovat tak slova, která jsou významově podobná.

Praktická část se nejprve věnovala tvorbě množiny webových stránek. Dále se pak věnovala návrhu a implementaci experimentálního softwaru, který klasifikuje webové stránky na vstupu programu do pěti různých kategorií, jedná se o kategorie, které se věnují psům, kočkám, gastronomii, fotbalu a ostatnímu obsahu. Významnou roli v tomto softwaru hraje nástroj *Selenium*, které slouží ke stahování webových stránek, a to jak dynamických, tak rámcových. Další významnou knihovnou je *Scikit-learn*. Díky této knihovně byly do programu naimplementovány klasifikační modely a technika TF-IDF, tou je následně realizována tvorba příznaků. Jako techniky pro předzpracování textových dat jsem zvolil tzv. lowering, odstranění šumu, odstranění stopwords a tokenizaci. Lemmatizace a stematizace v programu chybí z důvodů velmi omezené dostupnosti pro český jazyk. Klasifikace samotná pak probíhá na základně N-nejfrekventovanějších slov.

Součástí praktické části je i prezentace výsledků klasifikačních modelů. Testování bylo prováděno desetinásobnou křížovou validací a jako metrika byla zvolena celková přesnost modelu. Jelikož prvotní návrh systému počítal pouze se čtyřmi konkrétními kategoriemi, provedl jsem měření celkové přesnosti nejprve s těmito konkrétními kategoriemi a následně změřil celkovou přesnost za přítomnosti obecné kategorie *ostatní*. Tyto výsledky byly následně porovnány. Součástí výsledků je i měření celkové přesnosti pro různé argumenty klasifikátorů. Jako poslední byla zjišťována závislost N-nejfrekventovanějších slov, tedy jak ovlivní úspěšnost klasifikace změna parametru N. Při testování programu na jednotlivých stránkách jsem rovněž narazil na zajímavost,

která se týkala webu www.seznam.cz. Tento web obsahuje převážně odkazy na články aktuálního dění. V době testování se psalo hlavně o politice a program tedy vyhodnotil výstup jako kategorii *ostatní*. Párkrát se ovšem stalo, že aktuálním tématem byl fotbalový klub SK Slavia Praha a v těchto případech program vyhodnotil výstup jako kategorii *fotbal*.

Nedostatkem mého návrhu je pak především neřešení synonym a významově podobných slov. Z tohoto hlediska by opravdu pomohla implementace Word2Vec místo použitého TF-IDF, bohužel, jak jsem zmínil v teoretické části, pro klasické modely neexistuje žádné rozumné řešení, jak implementace dosáhnout. Jediná možnost jsou momentálně agregační funkce, při kterých ovšem také dochází ke ztrátě těchto detailů. Do budoucna, v případě navázání na tuto práci, mám v plánu konstrukci neuronové sítě, která by například mohla přijímat slova zakódovaná pomocí Word2Vec, a to přímo na vstupní vrstvě.

6 Seznam literatury, obrázků, tabulek a grafů

6.1 Literatura

[1] PANCHAL, Shubham. K Nearest Neighbor Classifier (kNN)-Machine Learning Algorithms. *Medium: equipintelligence* [online]. Mumbai, Maharashtra, India: https://equipintelligence.medium.com/, 2018 [cit. 2021-04-04]. Dostupné z: https://equipintelligence.medium.com/k-nearest-neighbor-classifier-knn-machine-learning-algorithms-ed62feb86582

[2] CHATTERJEE, Marina. A Quick Introduction to KNN Algorithm. *Great Learning* [online]. Great Learning, 2020 [cit. 2021-04-03]. Dostupné z: https://www.mygreatlearning.com/blog/knn-algorithm-introduction/

[3] KRČMÁŘ, Martin. *Analýza dat z výrobního procesu*. Technická 3082/12, Brno, Česká Republika, 2014. Bakalářská práce. Vysoké učení technické v Brně, Ústav automatizace a měřicí techniky. Vedoucí práce Prof. Ing. František Zezulka, CSc.

[4] BERKA, Petr. Dobývání znalostí z databází: Bayesovská klasifikace. *VSE: sorry* [online]. Praha: Vysoká škola ekonomická, 2006 [cit. 2021-04-04]. Dostupné z: https://sorry.vse.cz/~berka/docs/izi456/kap_5.6.pdf

[5] GABAŠOVÁ, Evelina. *Text Clustering and Classification*. Praha, 2007. Dostupné také z: https://is.cuni.cz/webapps/zzp/detail/46458/. Bakalářská práce. Univerzita Karlova v Praze. Vedoucí práce Mgr. Marta Vomlelová, Ph.D.

[6] DEISENROTH, Marc, A. FAISAL a Cheng ONG. *Mathematics for Machine Learning*. 1st edition. Cambridge University Press: Cambridge University Press, 2020. ISBN 110845514X.

[7] Support Vector Machines (SVM) (Algoritmy podpůrných vektorů): Přednáška. *Informační systém MUNI* [online]. Brno: Masarykova Univerzita, 2005 [cit. 2021-04-08]. Dostupné z: https://is.muni.cz/el/1433/podzim2006/PA034/09_SVM.pdf?fakulta=1433;obdobi=3

[8] STARMER, Josh. Introduction to Support Vector Machine. *AndreaPerlato.com* [online]. Basel Metropolitan Area: Perlato, 2019 [cit. 2021-04-08]. Dostupné z: https://www.andreaperlato.com/theorypost/introduction-to-support-vector-machine/

[9] Support Vector Machines: Kernels. In: *Scikit learn: documentation* [online]. c2007-2020 [cit. 2021-04-08]. Dostupné z: https://scikit-learn.org/stable/modules/svm.html#svm-kernels

[10] FRYDRYCHOVÁ, Alena. *PRAKTICKÉ MOŽNOSTI ANALÝZY TEXTOVÝCH DAT SE ZAMĚŘENÍM NA ANALÝZU SENTIMENTU A VIZUALIZACI* [online]. Arna Nováka 1/1, 602 00, Brno, 2020 [cit. 2021-04-09]. Dostupné z: https://is.muni.cz/th/n38t2/diplomova_prace.pdf. Diplomová práce. Masarykova Univerzita v Brně, Filozofická fakulta. Vedoucí práce Mgr. Tomáš Marek.

[11] SKLENÁK, Vilém, Daniela TKAČÍKOVÁ, Josef SCHWARZ a Jarmila BURGETOVÁ. Databáze Národní knihovny: KTS. In: *Národní knihovna České republiky: Databáze Národní knihovny* [online]. Praha: Databáze Národní knihovny, 2014 [cit. 2021-04-09]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000664&local_base=ktd

[12] HLADKÁ, Zdeňka a Václav CVRČEK. Lemma. In: *Petr Karlík, Marek Nekula, Jana Pleskalová (eds.), CzechEncy - Nový encyklopedický slovník češtiny.* [online]. Brno: CzechEncy, 2017 [cit. 2021-04-09]. Dostupné z: https://www.czechency.org/slovnik/LEMMA

[13] CVRČEK, Václav a Olga RICHTEROVÁ. Pojmy:lemma. In: *Příručka ČNK* [online]. Příručka ČNK., 2020 [cit. 2021-04-09]. Dostupné z: http://wiki.korpus.cz/doku.php?id=pojmy:lemma&rev=1608201720

[14] MATERNA, Jiří. Word2vec – simple word arithmetic. In: *Machine Learning Guru* [online]. Materna, 2015 [cit. 2021-04-10]. Dostupné z: http://www.mlguru.com/word2vec-jednoducha-aritmetika-se-slovy/

[15] SHAO, Taihua, Honghui CHEN a Wanyu CHEN. Query Auto-Completion Based on Word2vec Semantic Similarity. *Journal of Physics: Conference Series* [online].

2018, **1004**, 7 [cit. 2021-04-10]. ISSN 1742-6588. Dostupné z: doi:10.1088/1742-6596/1004/1/012018

[16] DE BOOM, Cedric, Steven VAN CANNEYT, Thomas DEMEESTER a Bart DHOEDT. Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recognition Letters* [online]. 2016, **80**, 150-156 [cit. 2021-04-10]. ISSN 01678655. Dostupné z: doi:10.1016/j.patrec.2016.06.012

[17] DIAS, Gene. Stopwords-cs: stopwords-iso. In: *GitHub* [online]. GitHub, 2020 [cit. 2021-04-17]. Dostupné z: https://github.com/stopwords-iso/stopwords-cs

[18] Feature extraction: scikit-learn. In: *Scikit-learn.org* [online]. Scikit-learn, 2020 [cit. 2021-04-18]. Dostupné z: https://scikit-learn.org/stable/modules/feature_extraction.html

[19] Abc: Abstract base classes. In: *Python: documentation* [online]. The Python Software Foundation, c2001-2021 [cit. 2021-04-18]. Dostupné z: https://docs.python.org/3/library/abc.html

[20] SVC: svm. In: *Scikit-learn.org: documentation* [online]. Scikit-learn, 2020 [cit. 2021-04-20]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[21] MIKOLOV, Tomas, Quoc LE a Ilya SUTSKEVER. Exploiting Similarities among Languages for Machine Translation. *ArXiv* [online]. 2013, **13094168**(13094168), 12 [cit. 2021-04-10]. Dostupné z: https://arxiv.org/abs/1309.4168

[22] LE, Quoc a Tomas MIKOLOV. Distributed Representations of Sentences and Documents. *ArXiv* [online]. 2014, **14054053**(140540532), 3-4 [cit. 2021-04-10]. Dostupné z: https://arxiv.org/abs/1405.4053

[23] REN, Qiubing, Mingchao LI a Shuai HAN. Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives. *Big Earth Data* [online]. 2019, **3**(1), 8-25 [cit. 2021-04-15]. ISSN 2096-4471. Dostupné z: doi:10.1080/20964471.2019.1572452

[24] Cross-validation: evaluating estimator performance. In: *Scikit-learn.org* [online]. Scikit-learn, 2020 [cit. 2021-04-18]. Dostupné z: https://scikit-learn.org/stable/modules/cross_validation.html

6.2 Seznam obrázků

6.3 Seznam tabulek

6.4 Seznam zdrojových kódů

6.5 Seznam grafů

7 Přílohy

Součástí této práce je přílohová složka, která je uložená ve školním systému na adrese www.wstag.jcu.cz. Tato složka obsahuje kompletní zdrojové kódy programu včetně souborů, které s tímto programem souvisí, dále obsahuje elektronickou verzi bakalářské práce.

Struktura

Patrik_Dohnal_2021_BP_prilohy.zip

```
README
    README.txt              # soubor s návodem na spuštění, dokumentací, užitím

Classifier.py
DatasetManager.py           # třídy programu
ProgramControl.py
ScrapeHandler.py
main.py                     # spouštěcí script
corpus_top_words.csv        # korpus N-nejfrekventovanějších slov
czech_stopwords.txt         # české stopwords
progressedData.txt          # data stažených stránek
progressedDataTemp.txt      # data stránek nově přidané kategorie
kneighbors_results.txt
naive_bayes_results.txt     # výsledky testů po naučení modelů
svm_results.txt
kat_psi.html
kat_kocky.html
kat_gastro.html             # vyexportované kategorie z prohlížeče
kat_fotbal.html
kat_ostatni.html
kNN_model.joblib
mNB_model.joblib            # serializované třídy, uložené modely
svm_model.joblib
```