Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Metody strojového učení pro analýzu textu

Jakub Šenk

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jakub Šenk

Informatika

Název práce

Metody strojového učení pro analýzu textu

Název anglicky

Machine learning methods for text analysis

Cíle práce

Cílem práce bude porovnání metod strojového učení pro analýzu a klasifikaci textu. V teoretické části budou představeny jednotlivé modely. V praktické části student na klasifikační úloze porovná a vyhodnotí tyto modely. Budou použita veřejně dostupná data, která hodnotí solventnost a finanční situaci pojišťoven a jsou analyzována regulátorem.

Metodika

Student popíše současný stav výzkumu v oblasti zpracování textu metodami strojového učení. V praktické části práce student na datech realizuje modely strojového učení pro klasifikační úlohu. Této úloze bude předcházet zpracování a optimalizace dat do strukturované podoby za použití běžných text miningových postupů. Použité modely strojového učení budou porovnány a bude vyhodnocena jejich efektivita. Kód bude psán v jazyku Python na platformě Jupyter Notebook, za použití knihoven jako jsou Scikit-learn, TensorFlow, Keras a NLTK.

Doporučený rozsah práce

60

Klíčová slova

Strojové učení, analýza textu, klasifikace

Doporučené zdroje informací

GÉRON, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Beijing ; Boston ; Farnham ; Sevastopol ; Tokyo: O'Reilly, 2019. ISBN 978-1-4920-3264-9.

CHOLLET, F. Deep learning v jazyku Python, Grada, Praha, 2019, ISBN 978-80-247-3100-1

Předběžný termín obhajoby

2022/23 ZS – PEF

Vedoucí práce

doc. Ing. Arnošt Veselý, CSc.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 4. 11. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 04. 03. 2023

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Metody strojového učení pro analýzu textu" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.3.2023 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Arnoštu Veselému, CSc. za odborné vedení a cenné rady při zpracování této práce. Také bych rád poděkoval kolegům a vedoucímu v České národní bance odboru dohledové statistiky za možnost zpracování této práce a poskytnutí podkladových dat.

Metody strojového učení pro analýzu textu

Abstrakt

Tato diplomová práce se zabývá analýzou textu metodami strojového učení. Cílem práce je sestavit a porovnat modely strojového učení s použitím různých klasifikačních algoritmů. V teoretické části práce jsou vysvětleny jednotlivé algoritmy, následně proces předzpracování textu a vektorizační techniky. Konec teoretické části ukazuje různé validační přístupy a hodnotící metriky. Praktická část práce začíná popisem datového souboru a přípravou dat. Datový soubor tvoří dokumenty zabývající se solventností a finanční situací pojišťoven. Dále jsou v praktické části práce sestaveny jednotlivé modely strojového učení, které jsou také vyladěny za použití GridSearch metody. Jsou použity různé techniky rozdělení datové sady a také různé přístupy předzpracování textu. V kapitole výsledky a diskuse jsou vyhodnoceny tyto rozdílné přístupy a jsou porovnány úspěšnosti jednotlivých modelů. Rovněž jsou zmíněny nedostatky řešení a návrh na zlepšení a pokračování dané úlohy. K práci je rovněž přiložen Python kód jednotlivých částí úlohy.

Klíčová slova: strojové učení, analýza textu, binární klasifikace, zpracování přirozeného jazyka, Python, scikit-learn

Machine learning methods for text analysis

Abstract

This diploma thesis deals with text analysis using machine learning methods. The goal of the thesis is to build and compare machine learning models using different classification algorithms. In the theoretical part of the thesis, individual algorithms are explained, followed by the process of text preprocessing and vectorization techniques. The end of the theoretical part describes validation approaches and evaluation metrics. The practical part of the thesis begins with the description of the dataset and data preparation. The dataset consists of documents dealing with the solvency and financial situation of insurance companies. Afterwards, individual machine learning models are built and also fine-tuned using the GridSearch method. Various dataset partitioning techniques are used as well as different text preprocessing approaches. In the conclusion and discussion chapter, these different approaches are evaluated and the accuracy rates of individual models are compared. Limitations of the solution and a suggestion for improvement and further processing of the work are also discussed. Python code for each part of the task is attached at the end of the thesis.

Keywords: machine learning, text analysis, binary classification, natural language processing, Python, scikit-learn

Obsah

1 Úvod

Umělá inteligence v poslední době prochází velkým vývojem a to se promítá i do oblasti analýzy textu. Analýza textu pomocí umělé inteligence se zabývá automatickým rozpoznáváním vzorů a trendů v textových datech pomocí různých metod. Cílem této analýzy může být například klasifikace textu do určitých témat, extrakce informací z textu, generování textu nebo rozpoznávání jazyka.

Největším průlomem v oblasti zpracování přirozeného jazyka (NLP, Natural language processing) bylo v poslední době představení chat bota ChatGPT firmou OpenAI. Model verze GPT-3 byl představen koncem roku 2022 a ihned se stal fenoménem, který je mnohými obdivován pro to, co všechno dokáže a pro některé obavy vyvolávající, jaké všechny profese tento nebo jiný model umělé inteligence dokáže v budoucnu nahradit. GPT-3 je velký jazykový model, který je schopen generovat velmi přesný a přirozený text v několika jazycích. Kromě odpovědí na klasické otázky a běžnou konverzaci je schopen také psát úvahy, řešit slovní úlohy nebo programovat. Je natrénován na velkém množství historických dat a kromě neuronových sítí využívá zpětnovazebního učení.

V této práci jsou popsány algoritmy strojového učení a dále je popsán postup pro použití metod strojového učení na textová data. Samotné zpracování a příprava textových dat do strojově zpracovatelné podoby je důležitá část u úloh strojového učení, proto jim je věnována podstatná část této práce.

K programování metod strojového učení je použit programovací jazyk Python a vývojové prostředí JupyterLab. Tato kombinace je velmi populární a často používaná v oblasti strojového učení a analýzy dat. JupyterLab je interaktivní vývojové prostředí, které umožňuje vytvářet dokumenty obsahující kód, vizualizace a text. Mezi jeho výhody patří mimo jiné spouštění kódu po částech v jednotlivých buňkách (cells). Tento nástroj je velmi vhodný pro rychlé prototypování a testování různých metod strojového učení.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem této diplomové práce je navrhnout a porovnat modely strojového učení pro analýzu a klasifikaci textu. V teoretické části je cílem představit jednotlivé modely a běžné postupy předzpracování textu. V praktické části je cílem na datech sestavit jednotlivé modely a vyhodnotit jejich efektivitu. Budou použita veřejně dostupná data, která hodnotí solventnost a finanční situaci pojišťoven a jsou analyzována regulátorem.

2.2 Metodika

V teoretické části práce jsou popsány algoritmy strojového učení pro klasifikaci textu. Dále je popsán proces zpracování textových dat za účelem použití metod strojového učení. Praktická část práce se zaměřuje na realizování modelů strojového učení pro klasifikační úlohu na textových datech. Použité modely strojového učení jsou porovnány a je vyhodnocena jejich efektivita. Kód je psán v jazyku Python na platformě JupyterLab. K importování a předzpracováni textu jsou použity knihovny pdfplumber, NumPy, pandas, matplotlib a NLTK. Vytváření modelů strojového učení je za použití knihoven scikit-learn, Keras a TensorFlow.

3 Teoretická část práce

3.1 Umělá inteligence

Umělá inteligence je jednoduše řečeno využití strojů a počítačů k napodobení lidské inteligence a k následnému vykonání zadaných úkolů. Je to univerzálním pojmem pro aplikace, které provádějí složité úkoly, které dříve vyžadovaly lidské vstupy. Umělá inteligence se v dnešní době využívá ve spoustě oblastech, ať už je to překlad jazyka, rozpoznání řeči, klasifikace obrazu, chatovací roboti, virtuální asistenti. (1)

Umělá inteligence je oblast, která zahrnuje strojové a hluboké učení, ale také další přístupy. Dříve byla umělá inteligence využívána programátory, kteří měli dostatečně velké soubory pravidel pro manipulaci s poznatky a vznikly tak expertní systémy, které jsou brány jako součást umělé inteligence. Tento přístup umožňoval řešit dobře definované logické problémy, jako je hraní šachů. Ovšem pro složitější problémy, které nemají daná pravidla, jako například rozpoznání řeči či obrazu, byl nedostačující. Proto vznikl nový přístup, strojové učení. (2, s. 21)

3.2 Strojové učení

Strojové učení je nedílnou součástí umělé inteligence. Ve strojovém učení jsou algoritmy trénovány pro nalezení vzorů a korelací v datech a pro provádění nejlepších rozhodnutí a predikcí na základě této analýzy. Úlohy strojového učení mohou být rozděleny podle struktury, množství a kvality dat, které máme k dispozici. Na základě dat a stanovení cíle se volí druh úlohy a algoritmus strojového učení. (3)

Dalo by se říci, že strojové učení řeší problémy matematickým modelováním vzorů v číselných sadách. Ve většině situací lze daný problém zredukovat na jeho číselné vyjádření. Například jednou z běžných aplikací pro strojové učení je analýza sentimentu. Příkladem může být text udávající recenzi restaurace, kde přiřazení 0 je pro negativní sentiment (například „jídlo bylo nevýrazné a služba byla hrozná.“) nebo 1 za pozitivní sentiment („Vynikající jídlo a služby. Nemůžu se dočkat, až vás znovu navštívím!“). Některé recenze mohou být smíšené, například „burger byl skvělý, ale hranolky byly

špatné“. V tomto případě je skóre sentimentu uvedeno pravděpodobností, že výsledné označení je 1. Velmi negativní komentáře mohou mít skóre 0,1, zatímco velmi pozitivní komentáře mohou mít skóre 0,9, protože existuje 90% šance, že vyjadřuje pozitivní sentiment. (4)

3.2.1 Učení s učitelem

Učení s učitelem (supervised learning) je nejběžnější případ strojového učení. Vyznačuje se tím, že trénovací data obsahují označenou cílovou proměnnou. Model strojového učení následně vytvoří algoritmus, na základě kterého je predikována cílová třída nebo hodnota v nových datech. Tato nová data již nemají označenou cílovou proměnnou.

Učení s učitelem lze rozdělit na dva základní druhy úloh - klasifikaci a regresi. V regresní úloze predikujeme numerickou hodnotu, cílová proměnná je spojitá. U klasifikace rozřazujeme cílovou proměnnou do dvou nebo více kategorií. V případě dvou kategorií hovoříme o binární klasifikaci, zatímco v případě více kategorií se jedná o klasifikaci do více tříd (multiclass). Typický případ binární klasifikace může být odchod zákazníka nebo predikce nemoci. Predikcí je pouze jedna ze dvou možností. V případě klasifikace do více tříd je dobrým příkladem rozpoznání jazyka, kdy výstupem je jeden jazyk z mnoha.

V některých situacích je potřebné mít možnost přiřadit vstupnímu záznamu více kategorií. V tomto případě hovoříme o klasifikaci s více výstupy nebo klasifikaci s více štítky (multilable). Například klasifikátor, který je natrénovaný na rozpoznání obličejů určitých lidí na fotce. Na každé fotce může tedy být buď žádný, jeden nebo více lidí. (5, s. 106)

3.2.2 Učení bez učitele

Učení bez učitele (unsupervised learning) je část strojového učení spočívající v hledání zajímavých transformací vstupních dat bez cílové proměnné pro účely vizualizace dat, komprese dat nebo lepšího pochopení korelací a souvislostí v daných datech. (2, s. 97)

Mezi úlohy učení bez učitele a algoritmy k jejich řešení patří:

- Shluková analýzy

    - Metoda nejbližších středů (K-Means)
    - Prostorové shlukování pomocí šumu (DBSCAN)
    - Hierarchické shlukování (HCA)
- Detekce anomálií
    - Izolační les (Isolation Forest)
    - Jednorozměrná metoda podpůrných vektorů (One-class SVM)
- Redukce dimenzionality
    - Analýza hlavních komponent (PCA)
    - Místní lineární vnoření (Locally Linear Embedding, LLE)
- Asociační a sekvenční úlohy
    - Apriori
    - Eclat

Speciální případ strojového učení je částečně řízené učení (semi-supervised learning). Tento druh úloh strojového učení vychází ze situací, kdy část dat jsou cílové proměnné označeny a zbytek nikoliv. Některé algoritmy si s tímto případem dokážou poradit, kdy jsou použity prvky z obou hlavních přístupů. (5, s. 13)

3.2.3 Zpětnovazební učení

Zpětnovazební učení (reinforcement learning) je druh strojového učení, který se více než kterýkoliv jiný snaží napodobit chování lidského mozku. Agent má za úkol dosáhnout nejlepších výsledků získáváním informací o předmětu zkoumání a snaží se poučit ze svých vlastních chyb tím, že volí akci, která maximalizuje odměnu. Zpětnovazební učení bylo použito společností Google DeepMind, která se zasloužila o její propagaci zejména tím, že vytvořila aplikaci, která se naučila hrát video hry a později stolní hru Go na té nejvyšší úrovni. Přes to všechno zpětnovazební učení je spíše na počátku vývoje a prozatím se v praxi používá málokdy. DeepMind také udává možnosti využití zpětnovazebního učení jako např. řízení helikoptéry, ovládání robotů nebo správa investičního portfolia. (6)

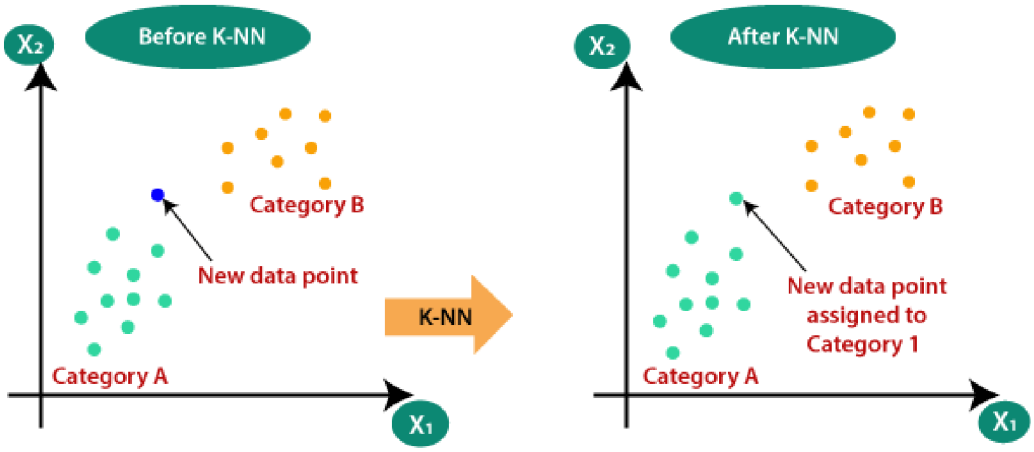
3.3 Algoritmy strojového učení

Každá z částí strojového učení má své algoritmy, které jsou vhodné k použití na daný typ úlohy. Některé algoritmy mohou mít využití ke klasifikaci, jiné zas k regresi. Některé z nich se lehkým rozšířením nebo úpravou mohou použít na obě tyto úlohy. Ovšem je jen málo případů, kdy lze využít algoritmus daný na problematiku učení s učitelem například ke zpětnovazebnímu učení nebo na úlohu učení bez učitele. V této kapitole budou detailněji probrány základní algoritmy na klasifikační úlohy.

3.3.1 K-nejbližších sousedů

Algoritmus K-nejbližších sousedů (K-nearest neighbors, KNN) je jeden z nejzákladnějších a nejjednodušších neparametrických algoritmů. Zjednodušeně řečeno KNN algoritmus předpokládá podobnost mezi novým případem a již dostupnými daty a klasifikuje je na základě vzdálenostní metriky. KNN je primárně klasifikační algoritmus, ale lze ho použít i na regresní úlohy. Tento algoritmus je vyhodnocován tzv. líným vyhodnocením, což znamená, že výpočet probíhá až ve chvíli, kdy je provedena predikce. (7)

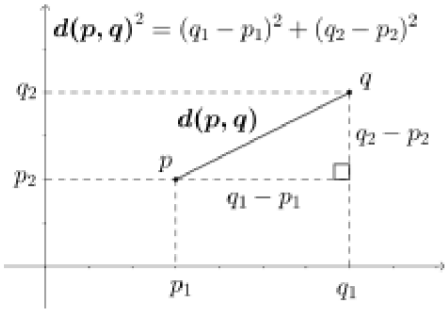
Obrázek 1 - KNN algoritmus



Zdroj: (7)

KNN je založen na principu vzdálenostních metrik. Nejběžnější metrika je Euklidovská vzdálenost, která se vypočítá jako druhá odmocnina ze součtu druhých mocnin rozdílu mezi novým bodem a existujícím bodem:

Obrázek 2- Euklidovská vzdálenost



Zdroj: (8)

Další populární metrikou je Manhattanská vzdálenost, která je dána absolutní hodnotou mezi dvěma body. Tato metrika je inspirována automobilem New Yorské taxi služby, který musí urazit co nejkratší vzdálenost mezi dvěma křižovatkami. (9) Manhattanská vzdálenost má následující tvar:

$$\sum_{i=1}^{m} |p_i - q_i|$$

Na základě předchozích dvou metrik je definována Minkowského vzdálenostní metrika $\left(\sum_{i=1}^{m} |x_i - y_i|^p\right)^{1/p}$, kde

P = 1 Manhattan distance

P = 2 Euclidean distance

Obecný postup výpočtu KNN klasifikátoru by se mohl skládat z následujících kroků:

1. Zvolit počet K sousedů.
2. Zvolit vzdálenostní metriku a vypočítat vzdálenost ke K sousedům.
3. Ze všech K nejbližších sousedů spočítat kolik patřilo do jaké kategorie.
4. Přiřadit nová data do kategorií, která byla obsažena nejvíce.

Mezi výhody KNN klasifikátoru patří především lehká implementace a odolnost proti lehce poškozeným trénovacím datům. Nevýhodami jsou především nutnost manuálně nastavovat hodnotu K a náročnost na výpočetní výkon. Pro KNN algoritmus je vhodné mít velké množství trénovacích dat a malé množství vstupních proměnných. (7)

3.3.2 Metoda podpůrných vektorů

Metoda podpůrných vektorů (Support Vector Machine, SVM) je algoritmus primárně pro klasifikační úlohy, avšak má uplatnění i v regresním modelování. Princip SVM je rozdělit množinu bodů přímkou nebo nadrovinou do jednotlivých kategorií. Pokud je vstupní množina dat lineárně separabilní, používá se obyčejný lineární SVM model. Mnoho datasetů ovšem není lineárně separabilních. V tomto případě lze využít polynomiálních proměnných (polynomial features), které vytváří nové proměnné na základě pronásobením již existujících proměnných a to může vést ke vzniku lineárně separabilní množiny. Pro stupeň polynomu 2 a proměnné [a,b] vznikají nové proměnné následujícím způsobem (10): [1, a, b, a^2, ab, b^2]

Přidání polynomiálních proměnných lze využít i u jiných algoritmů strojového učení. Ovšem v některých případech, zvláště pokud je třeba použít polynom velkého stupně, vznikne datový soubor s velkým počtem proměnných, což zpomaluje model a vytváří ho náročným na výpočetní výkon.

Proto se využívá SVM s jiným než lineárním jádrem. K tomu je určený tzv. jádrový trik (kernel trick), díky kterému se provádí transformace dat z původního prostoru do prostoru vyšší dimenze, ve kterém jsou data již lineárně separabilní. Další nejpoužívanější jádra, kromě klasické lineárního jsou (11):

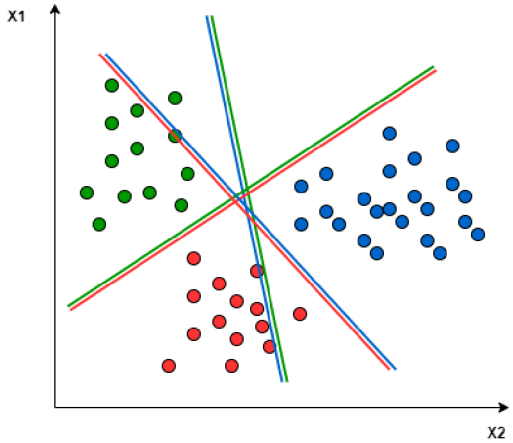
- **Polynomické jádro**, které je rozšířením jádra lineárního, je dáno vztahem: $F(x, x_j) = (x.x_j + 1)^d$, kde F(x, x_j) reprezentuje rozdělovací přímku pro oddělení tříd a d značí stupeň polynomu.
- **RBF jádro (Gaussian Radial Basis Function)**, které bývá tím vůbec nejpoužívanějším pro nelineární data. Je dáno vztahem: $F(x, x_j) = \exp\left(-gamma * \left||x - x_j|\right|^2\right)$, kde ||x - x_j|| je euklidovská vzdálenost mezi body x a x_j. Gamma je hyperparametr v intervalu od 0 do 1.
- **Sigmoidní jádro**, je jádro podobné funkce jako dvouvrstvý model perceptronu, který se používá jako aktivační funkce v neuronových sítích. Sigmoidní jádro je dáno vztahem:

$F(x, x_j) = \tanh(\alpha x^T x_j + c_0)$, kde x a y jsou vstupními vektory, α je sklon (slope), c_0 je výchozí hodnota (intercept)

SVM algoritmus ve své podstatě poskytuje pouze binární druh klasifikace, ovšem lze ho využít i ke klasifikaci do více tříd rozložením na několik binárních klasifikací. K tomu se využívají dva přístupy:

One-to-One přístup je založený na oddělení každých dvou vzájemných tříd navzájem, bez ohledu na třídy ostatní. Vzniká tedy binární klasifikátor pro každou dvojici tříd. V tomto případě vzniká pro *m* tříd $\frac{m(m-1)}{2}$ binárních klasifikátorů. Pro každý vstupní záznam, je potřeba spustit všechny klasifikátory a vyhodnotit, ke které třídě se záznam přiřazuje nejčastěji.

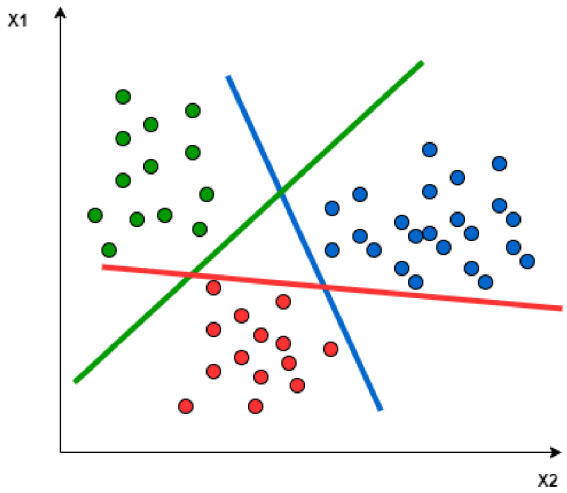
Obrázek 3 - One-to-One SVM klasifikace



Zdroj: (12)

Druhým přístupem je One-to-Rest, kde je každá třída rozdělena proti zbytku všech tříd, jak kdyby se jednalo o jednu velkou třídu. Znázornění One-to-Rest klasifikace do tří tříd je na obrázku 4. Je tedy vytvořeno tolik klasifikátorů, kolik je tříd. Pokud chceme přiřadit záznam do jedné z tříd vybere se ta třída, která měla nejvyšší skóre oproti zbytku. (12)

Obrázek 4 - One-to-Rest SVM klasifikace



Zdroj: (12)

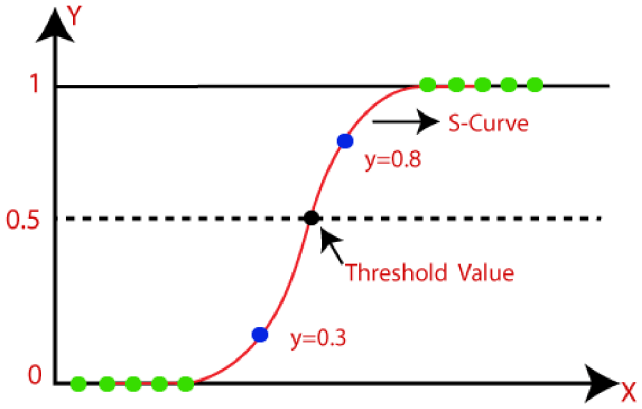
SVM algoritmus je vhodným klasifikátorem pro menší a středně velké datové soubory, avšak je dobrý pro soubory s velkým počtem dimenzí, dokonce je vhodným pro modelování datových souborů, kde je větší počet dimenzí než záznamů.

3.3.3 Logistická regrese

Logistická regrese je klasifikační algoritmus, který počítá pravděpodobnostní zařazení dat do tříd. Pokud odhad pravděpodobnosti je vyšší než 50 %, je záznam přiřazen k této třídě. Logistická regrese je tedy binární klasifikátor. Algoritmus logistické regrese vychází ze sigmoidální neboli logistické funkce (5, s. 142):

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

Obrázek 5 - Logistická funkce pro klasifikaci



Zdroj: (13)

Predikce zařazení do tříd „1“ a „0“ je dána následujícím vztahem:

$$y = 0 \; if \; p < 0.5$$
$$y = 1 \; if \; p \geq 0.5$$

Skóre *t* se často nazývá logit. Název pochází ze skutečnosti, že funkce logit, definovaná jako $logit(p) = \log(p \,/\, (1 - p))$, je inverzní k logistické funkci. Logit se také nazývá log-odds, protože je logaritmem poměru mezi odhadovanou pravděpodobností pro kladnou třídu a odhadovanou pravděpodobností pro negativní třídu. (5, s. 144)

Logistická regrese patří mezi lehce implementované a interpretovatelné algoritmy. Lepších výsledků dosahuje v případě lineárně separabilní množiny dat, protože je dána lineární rozhodovací plochou. Logistická regrese vyžaduje, aby nebyla multikolinearita mezi nezávislými proměnnými, ale zároveň předpokládá závislost mezi nezávislými a cílovou proměnnou. V případě klasifikace do více tříd je možné použít Softmax regresi nebo multinomiální logistickou regresi. (13)

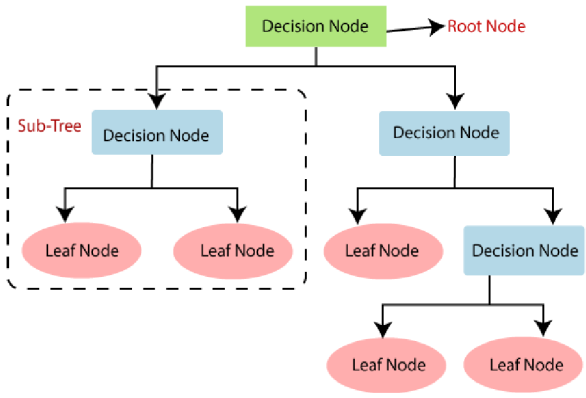
3.3.4 Rozhodovací stromy

Rozhodovací strom je technika učení s učitelem, kterou lze použít jak pro klasifikační, tak pro regresní problémy, avšak většinou je preferována pro řešení klasifikačních problémů. Jedná se o stromově strukturovaný klasifikátor, kde vnitřní uzly představují vlastnosti

datové sady, větve představují rozhodovací pravidla a každý listový uzel představuje výsledek.

V rozhodovacích stromech jsou dva druhy uzlů. Je to rozhodovací uzel a listový uzel. Rozhodovací uzly se používají k rozhodování a mají více větví, zatímco listové uzly (někdy jen listy) jsou výstupem těchto rozhodnutí a neobsahují žádné další větve. Jedná se o grafické znázornění pro získání všech možných řešení rozhodnutí na základě daných podmínek. (14)

Obrázek 6 - struktura rozhodovacího stromu



Zdroj: (14)

Rozhodovací stromy používají více algoritmů k rozhodnutí, zdali rozdělit uzel na dva nebo více pod uzlů a do jaké úrovně hierarchie stromové struktury přiřadí daný atribut. Pokud se datová sada skládá z mnoha atributů, je rozhodnutí, který atribut umístit do kořenového uzlu nebo na jakou úroveň stromu, komplikovaným krokem. Pouhým náhodným výběrem libovolného uzlu jako kořenového adresáře nelze problém vyřešit. Pokud použijeme náhodný přístup, může nám to poskytnout špatné výsledky s nízkou přesností. Pro vyřešení tohoto problému se využívají kritéria na základě kterých se provádí tato rozhodnutí. (15)

- **Entropy** - Míra náhodnosti ve zpracovávaných informacích. Čím vyšší je entropy, tím těžší je z této informace vyvodit nějaké závěry. Hození mincí je příkladem

akce, která poskytuje informace, které jsou náhodné. Entropy pro jeden atribut je dána vztahem:
$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i$, S je konkrétní atribut a p_i je pravděbodobnost třídy i.

- **Informační zisk (Information gain)** - Informační zisk je statistická vlastnost, která měří, jak dobře daný atribut odděluje tréninkové případy vzhledem k jejich cílové klasifikace. Konstrukce rozhodovacího stromu je o nalezení atributu, který vrací nejvyšší informační zisk a nejmenší entropii.

$$Informační\ zisk = Entropy(před) - \sum_{j=1}^{K} Entropy(j, po)$$

, kde „před“ značí atribut před rozdělením, K je počet rozdělených uzlů a (j, po) značí daný uzel atributu po rozdělení.

- **Gini Index -** Gini Index, vypočítává míru pravděpodobnosti specifického prvku, který je při náhodném výběru klasifikován nesprávně. Pokud jsou všechny prvky spojeny s jedinou třídou, lze ji nazvat čistou**.** Giniho index je určen rozdílem součtu mocnin pravděpodobností každé třídy od jedné. Matematicky lze Giniho index vyjádřit jako:

$$Gini\ Index = 1 - \sum_{i=1}^{c} (p_i)^2$$

- **Informační poměr (Gain ratio) -** Informační zisk má sklon vybírat atributy s velkým počtem hodnot jako kořenový uzel. Gain ratio překonává tento problém tím, že bere v úvahu počet větví, které by vznikly před provedením rozdělení.

$$Informační\ poměr = \frac{Informační\ zisk}{Entropy}$$

Jednotlivé algoritmy rozhodovacích stromů využívají zejména tato kritéria. Mezi nejznámější a nejpoužívanější algoritmy patří:

ID3

Algoritmus ID3 staví rozhodovací stromy pomocí přístupu „top-down greedy search approach“ bez možnosti zpětného sledování. Jak název napovídá, vždy je vybrán atribut, který se v danou chvíli zdá být nejlepší a zařadí jej do daného uzlu. Větev s nulovou entropy je listový uzel a větev s entropy větší než nula potřebuje další rozdělení. Vybírá tedy atributy s nejmenší entropy mírou nebo alternativně atributy s nejvetším informačním ziskem. (15)

C4.5

C4.5 je algoritmus, který rozšiřuje ID3 několika vylepšeními. Umí pracovat se spojitými hodnotami tím, že vytvoří hranici a rozdělí tuto proměnnou na binární kategorickou. Lépe dokáže zpracovávat nulové hodnoty, které označí znakem „*?*“ a následně je vynechává při výpočtu metrik. C4.5 na rozdíl od ID3 využívá zpětného sledování, kdy po vytvoření stromu jde zpět a odstraňuje větve, které mají malý přínos.

CART

Classification and Regression Trees (CART) je binární algoritmus, který rozděluje data hierarchicky do podskupin s nižší variabilitou. Rovněž podporuje penalizaci chyb a prořezávání stromu. Pro diskrétní atributy hledá vhodné sloučení kategorií do dvou skupin a pro spojité rozdělí hodnoty na dva intervaly.

CHAID

Chi – square Automatic Interaction Detection (CHAID) je nebinární strom určen pro kategorické proměnné. Tento algoritmus provádí tři hlavní kroky: slučování kategorií, štěpení uzlu a ukončení růstu. Pro každý pár kategorií, které lze sloučit určí sílu závislosti s cílovou proměnnou. K tomu používá chi-square hodnotu pro klasifikační úlohu a F-test pro regresní úlohu. Vybere se pár s nejvyšší dosaženou hladinou významnosti *p* a pokud $p>\alpha_{merge}$, pak jsou kategorie sloučené v jedinou. Následuje štěpení uzlů, kde se vybírá prediktor s nejnižší dosaženou hladinou významnosti *p* a pokud $p<\alpha_{split}$ tak je vytvořen dceřiný uzel. Tento postup se opakuje, dokud není možnost dalšího štěpení. (16)

3.3.5 Naivní bayes

Naivní Bayes je jedním z jednoduchých a účinných klasifikačních algoritmů, který slouží k vytváření rychlých modelů strojového učení. Je založených na aplikaci Bayesova teorému s „naivním“ předpokladem, že všechny proměnné jsou mezi sebou nezávislé. Bayesův teorém se používá k určení pravděpodobností hypotézy s předchozími znalostmi. Vzorec pro Bayesův teorém je (17):

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

,kde $P(A|B)$ je podmíněná pravděbodobnost jevu A za předpokladu jevu B, $P(B|A)$ je opačná podmíněná pravděpodobnost, $P(A)$ je $P(B)$ jsou pravděpodobnosti jevu A a B.

Mezi 3 hlavní druhy Bayesového modelu patří:

- **Gaussův**
  Gaussův model předpokládá, že proměnné mají normální rozdělení. To znamená, že pokud prediktory nabývají spojitých hodnot namísto diskrétních, pak model předpokládá, že tyto hodnoty jsou vzorkovány z Gaussova rozdělení. (17)

- **Multinomiální**
  Multinomiaální Bayesovský klasifikátor se používá, když jsou data distribuována multinomiálně. Primárně se používá pro problémy s klasifikací dokumentů, to znamená, že konkrétní dokument patří do jedné z kategorií, jako je sport, politika, vzdělávání atd.

- **Bernoulliho**
  Bernoulliho klasifikátor se používá pro data, která mají alternativní Bernoulliho rozdělení, kde prediktorové proměnné jsou nezávislé booleovské proměnné. Například, zda je určité slovo v dokumentu přítomno nebo ne. Tento klasifikátor může u některých datových sad fungovat lépe, zejména u těch s kratšími dokumenty. (18)

Navzdory svým zjevně příliš zjednodušeným předpokladům fungují naivní Bayesovy klasifikátory docela dobře v mnoha situacích reálného světa, například klasifikace dokumentů a filtrování spamu. K odhadu potřebných parametrů vyžadují malé množství tréninkových dat. (18)

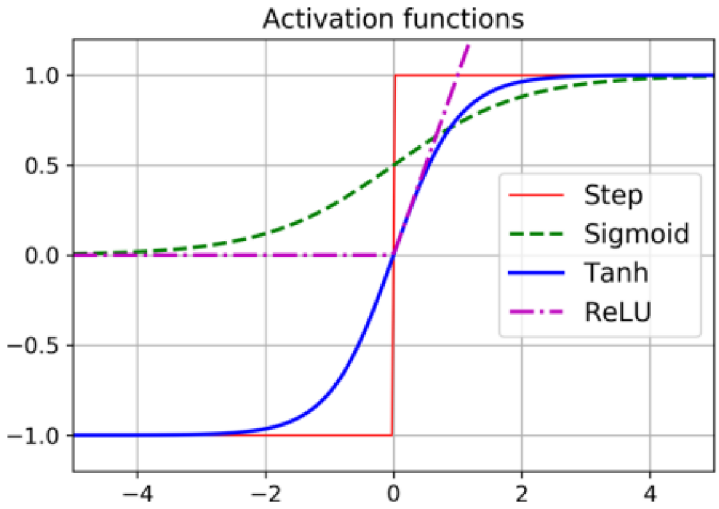
Mezi výhody tohoto klasifikátoru patří možnost použití pro binární klasifikace i klasifikace do více tříd, kde ve srovnání s ostatními algoritmy funguje poměrně dobře. Rychlost trénování patří i díky své jednoduchosti k nejméně náročným. Mezi nevýhody patří předpoklad, že všechny proměnné jsou nezávislé.

3.3.6 Neuronové sítě

Neuronové sítě jsou algoritmy modelované podle biologické aktivity lidského mozku. Skládají se z uzlů, nazývaných také neurony. Kolekce vertikálních uzlů se nazývá vrstva. Model se skládá z jednoho vstupu, jednoho výstupu a několika skrytých vrstev.
Uzly jedné vrstvy jsou spojeny s další a komunikují spolu pomocí signálů. Každý neuron přijímá vstupy z jiných neuronů nebo ze vstupních dat a generuje výstupní signál, který ovlivňuje další neurony v síti.

O tom, zda má být neuron aktivován nebo ne rozhoduje aktivační funkce výpočtem váženého součtu a dalším přidáním určitého zkreslení. Tyto funkce přidávají nelinearitu do výstupu neuronů. Existuje mnoho různých typů aktivačních funkcí, které se používají v neuronových sítích. V dnešní době jedna z nejpoužívanějších je funkce ReLU a její varianty. ReLU se vyznačuje zejména malou náročností na výpočetní výkon. Dále sigmoidní funkce, která se používá na výstupní vrstvě u klasifikačních úloh. Tanh funkce má zas využití ve skrytých vrstvách nebo třeba softmax funkce, která se používá zejména na klasifikaci do více tříd. Každá funkce má své vlastní výhody a nevýhody a vybírá se na základě konkrétních požadavků modelu a datové sady. Aktivační funkce, včetně základní „step“ funkce jsou znázorněny na obrázku 7. (19)

Obrázek 7 - Aktivační funkce neuronových sítí



Zdroj: (5, s. 292)

Další důležitou součástí trénování neuronové sítě je optimalizátor. Jeho úkolem je minimalizovat chybu sítě a přizpůsobit váhy tak, aby byly nejefektivnější pro předpovídání vstupních dat. Váhy jsou parametry v jednotlivých vrstvách, na základě kterých probíhá transformace dat.

Optimalizátor se používá k optimalizaci funkcí pomocí gradientního sestupu. Gradientní sestup je algoritmus, který sleduje směr největšího poklesu výsledného výstupu sítě a upravuje váhy sítě tak, aby minimalizoval chybu. Nejznámějšími optimalizační algoritmy jsou stochastický gradientní sestup (SGD), Adagrad, RMSprop nebo Adam.

Cílem trénování neuronové sítě je minimalizovat hodnotu ztrátové funkce. To se obvykle provádí pomocí algoritmu zpětné propagace chyby, který vypočítává gradient ztrátové funkce vzhledem k vahám neuronové sítě a upravuje je tak, aby minimalizoval ztrátovou funkci. Ztrátové funkce se používají dle typu úlohy neuronové sítě. Například pro klasifikační úlohy se využívá cross–entropy loss, hinge loss, případně binary cross-entropy loss pro binární klasifikaci.

Nejpoužívanější druhy neuronových sítí a jejich použití jsou:

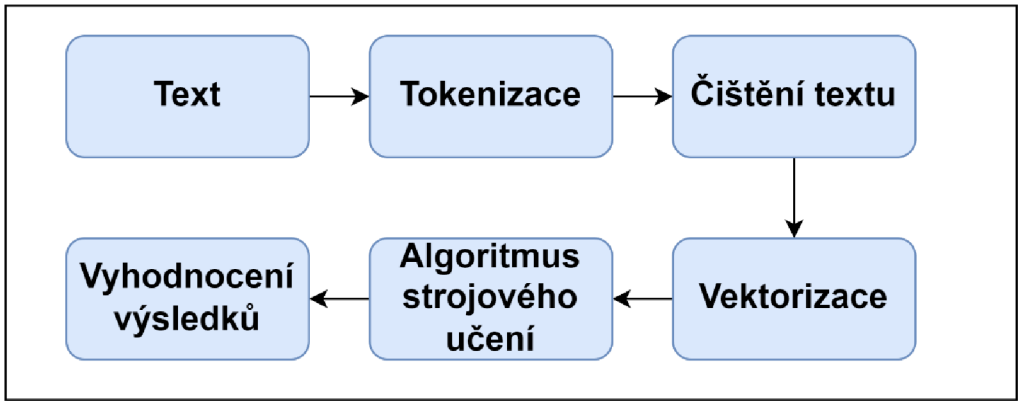
- Dopředné neuronové sítě – Rozpoznávání vzorů, počítačové vidění

- Vícevrstvý percepton – Strojový překlad, klasifikace
- Konvoluční neuronové sítě – Zpracování přirozeného jazyka, detekce anomálií
- Rekurentní neuronové sítě – Generování textu, Předpovídání vývoje akciových trhů
- LSTM (Long short – Term memory) – Rozpoznání řeči, rozpoznání rukopisu (20)

3.4 Proces předzpracování textu

Algoritmy strojového učení ve své podstatě nedokáží zpracovávat text a proto je třeba jej nejdříve transformovat do číselné podoby. Předzpracování textu zahrnuje zejména tokenizaci a čištění textu. Postup celého zpracování textu a vytvoření modelu strojového učení na textová data zahrnuje několik dalších kroků a je znázorněn na obrázku číslo 8.

Obrázek 8- Proces zpracování textu



Zdroj: Vlastní zpracování

3.4.1 Tokenizace

Tokenizace je jednoduchý proces, který bere nezpracovaná data a převádí je na lépe zpracovatelný datový řetězec. Tokenizace se používá při zpracování přirozeného jazyka k rozdělení odstavců a vět na menší jednotky, kterým lze snadněji přiřadit význam. Rozdělení vět na části umožňuje stroji porozumět částem textu i celku. To pomůže programu porozumět každému slovu samotnému a také tomu, jak fungují v celkovém

kontextu. Což je zvláště důležité pro větší množství vstupního textu, protože to umožňuje stroji spočítat frekvence určitých slov a také to, kde se často vyskytují. To je důležité pro pozdější kroky při zpracování přirozeného jazyka. Existuje několik druhů tokenizace, které zásadně ovlivňují další kroky v procesu předzpracování textu. (21)

Slovní tokenizace

Nejběžnější způsob vytváření tokenů je založen na základě mezery mezi slovy. Z toho vyplívá, že každý token je slovo. Nejznámějším způsobem je tedy slovní tokenizace (Word tokenization). Jedním z hlavních problémů se slovními tokeny je řešení slov „Out Of Vocabulary"(OOV). Slova OOV jsou nová slova, která se objevila při testování, avšak nebyla spatřena při trénování. Tato nová slova neexistují ve slovní zásobě, což může způsobovat značné problémy. Proto lze použít další přístupy.

Tokenizace znaků

Tokenizace znaků (Character tokenization) rozdělí část textu na sadu znaků a reprezentuje tak každé písmeno nebo znak zvláště. Minimalizuje se tím tedy nepřesnost OOV či případně špatně napsaná slova. Délka vstupního a výstupního tokenizovaného textu se tím pádem zvětšuje, protože větu reprezentujeme jako sekvenci znaků. V důsledku toho je náročné naučit se vztah mezi znaky, které tvoří smysluplná slova.

Tokenizace podslov

Tokenizace podslov (Subword tokenization) rozdělí část textu na kořeny slov nebo n-gram znaků. Široce používaná metoda tokenizace podslov je Byte Pair Encoding (BPE). BPE řeší problémy Slovní a znakové tokenizace. Segmentuje OOV jako podslova a reprezentuje slovo z hlediska jeho hlavní části. Délka vstupních a výstupních vět po BPE je kratší ve srovnání s tokenizací znaků. BPE je algoritmus segmentace slov, který iterativně spojuje nejčastěji se vyskytující znaky nebo sekvence znaků. (22)

Příklad jednotlivých tokenizací na větě „*Let´s learn tokenization.*" by vypadal následovně:

Slovní tokenizace: [*"Let´s", "learn", "tokenization"*]

Tokenizace znaků: [“L“, “e“, “t“, “$'$“, “s“, “l“, “e“, “a“, “r“, “n“,
“t“, “o“, “k“, “e“, “n“, “i“, “z“, “a“, “t“, “i“, “o“, “n“]

Tokenizace podslov: [“Let“, “$'$“, “s“, “$learn$“, “$token$“, “$ization$“]

3.4.2 Čištění textu

Čištění textu je proces transformace textu do lépe zpracovatelné formy. Díky tomu algoritmy strojového učení dosahují lepších výsledků. Čištění textu využívá různé techniky a může se skládat z několika kroků: Sloučení velikosti písmen, odebrání interpunkce, odstranění nepotřebných znaků a stop slov, stemming a lemmatizace.

Velikost písmen

V mnoha úlohách analýzy textu modely využívají objekt zvaný slovník (vocabulary). Slovník je soubor jedinečných slov v celém korpusu dokumentů. Normalizací všech slov na stejně velká písmena je dosaženo zredukování duplicit kvůli velikosti různých písmen. To má pozitivní vliv na velikost vstupního souboru, což snižuje požadavky na výpočetní výkon modelování.

Odebrání interpunkce

Ačkoliv interpunkce má v psaném textu velký význam, pro samotný algoritmus strojového učení nemá velký vliv a zařazením na vstup by mohla vnést do modelu určitý šum. Důvod odebrání interpunkce je především ten, že sama o sobě nemá žádný význam. Význam má pouze v kontextu celé věty nebo alespoň několika slov. Model strojového učení však dokáže pracovat pouze s jednotlivými slovy, případně s n-gramem po sobě následujících slov, proto odebrání interpunkce i s ohledem na velikost vstupní matice bývá vhodný krok přípravy dat. Existují však případy, kdy je žádoucí ponechat interpunkci a znaky ve svých datech. Například v úlohách generování textu může být užitečné zachovat interpunkci, aby daný model mohl generovat text, který je gramaticky správný.

Odstranění nepotřebných znaků a stop slov

Dalším krokem je odstranění všech znaků, které našemu dokumentu nepřidávají velkou hodnotu nebo význam. To kromě již zmíněné interpunkce může znamenat čísla,

emotikony, data a zejména stop slova (stop words). Při zpracování přirozeného jazyka je stop slovo jakékoli slovo, které větě nepřidává velký význam. Odstraněním těchto slov se zmenší velikost našeho slovníku, přičemž všechny relevantní informace v tomto dokumentu zůstanou zachovány. Ne vždy úlohy jazykového modelování považují za užitečné odstranit stop slova, například překlad nebo generování textu. Je to proto, že tyto úkoly stále berou v úvahu gramatickou strukturu každého dokumentu a odstranění určitých slov může vést ke ztrátě této struktury. V závislosti na úloze je důležité mít na paměti, která stop slova jsou z jazykového korpusu odstraněna. Stop slova bývají zejména předložky, spojky, částice, citoslovce a zájmena. Slovník stop slov je třeba definovat pro každý jazyk a každou úlohu zvlášť. (23)

Stemming a lematizace

Stemming a Lematizace jsou algoritmy, které se používají ve zpracování přirozeného jazyka k normalizaci textu. Cílem obou přístupů je zredukovat slova na jejich slovní kořen, případně na základní kontext daného slova. To má za následek, že stejná slova v jiném tvaru budou později zakódována stejně.

Stemming anglických slov „university“, „universal“ a „universe“ mají jako výsledek stemování slovní kmen „univers“, což přiřadí všechna tato slova ke stejnému významu. To v tomto případě není žádoucí, protože všechny tři slova mají jiný význam. Tomuto jevu se říká overstemming. Stemming tedy kóduje slova bez přihlédnutí k jejich významu. Opačným případem je problém understemming, kdy slova stejného významu jsou přiřazena jinému stemmu. (24)

Dalším problémem je, že jednotlivé stemmovací algoritmy jsou tvořeny pro konkrétní jazyky. Například velmi známý a jeden z nejpoužívanějších algoritmů PorterStemmer, který je populární pro svoji jednoduchost a rychlost, je pouze pro anglický jazyk. Dalším známým algoritmem pro anglický jazyk je LancasterStemmer, který je více agresivní, tzn. má tendenci jednotlivá slova stemmovat na kratší kořeny. Dalším zajímavým algoritmem a projektem je SnowballStemmer, který nabízí algoritmy pro mnoho dalších jazyků a je kompilován do mnoha programovacích jazyků, včetně Pythonu. Český jazyk bohužel mezi podporované jazyky nepatří. (25)

Lematizace, na rozdíl od stemmingu, správně redukuje skloňovaná slova a zajišťuje, že kořen slova má význam z hlediska lingvistiky. V lemmatizaci se kořen slova nazývá lemma. Lematizace je přesnější a při správném použití přináší lepší výsledky než stemming, ovšem za cenu větší výpočetní složitosti.

3.5 Vektorizace

Vektorizace je přístup k převodu vstupních dat z textového formátu do vektorů reálných čísel, což je formát, který modely strojového učení podporují. Jednotlivá čísla jsou zapsána ve formě matic či vektorů. Vektroizace je většinou aplikována na již očištěná data. Existuje několik přístupů vektorizace textu. (26)

3.5.1 Bag of words

Technika bag of words převádí textový obsah na vektory numerických znaků. Po provedení tokenizace se vybere každé slovo pouze jednou a seřadí se podle abecedy do slovníku. Následně se vytvoří prořídlá matice (sparse matrix), která má stejnou délku, jako je počet unikátních slov. Každá věta je reprezentována jedním řádkem, kdy „0“ značí nepřítomnost daného slova a kladné číslo značí počet výskytů daného slova, jak lze vidět na obrázku 9. Ve většině datových souborů bude každý řádek velmi dlouhý s velkým podílem „0“ hodnot, proto se této matici říká prořídlá. Další nevýhoda je, že nezachovává pořadí slov ve větách. (27)

Obrázek 9 - Ukázka bag of words tokenizace

Zdroj: (27)

Pro zachování alespoň částečného pořadí slov ve větách lze vytvořit slovník seskupených slov. To umožňuje zachytit z textu o něco více užitečných informací. V tomto přístupu se každé slovo nebo token nazývá gram. Vytváření slovníku dvouslovných dvojic se nazývá model bigramu. N-gram je n-tokenová posloupnost slov. Čím vyšší n-tokenová posloupnost slov je zvolena, tím více informace z pořadí slov ve větách lze získat, ovšem pouze za cenu vyšší výpočetní složitosti. Proto je vždy třeba zvolit vhodný kompromis. Princip N-gramů lze použít i u jiných přístupů vektorizace. (28) Příklad použití bigramu BoW pro větu *„analýza textu může mít velký přínos“:*
[[analýza textu] [mít velký] [může mít] [textu může] [velký přínos]]

3.5.2 TF-IDF

TF-IDF (Term Frequency – Inverse Document Frequency) je metoda, která má odrážet, jak důležité je slovo pro celý korpus. TF-IDF na rozdíl od bag-of-words bere v potaz, jak často se dané slovo vyskytuje v korpusu. Tato metoda považuje často se vyskytující slova jako méně důležitá.

První část TF vyjadřuje četnost slova v celém dokumentu a je dána vzorcem:

$$TF = \frac{Četnost\ slova\ n\ v\ dokumentu}{Celková\ počet\ slov\ v\ dokumentu}$$

Druhá část IDF počítá důležitost slova v korpusu, která je dána četností daného slova. Korpus je soubor všech dokumentů v rámci modelovací úlohy. V určitých případech může být korpus složen pouze z jednoho dokumentu a ten rozdělen na určité kapitoly. V tomto případě lze brát kapitolu jako jeden dokument. IDF je dána vzorcem (26):

$$IDF = \log\left(\frac{celkový\ počet\ dokumentů}{počet\ dokumentů\ obsahující\ slovo\ n}\right)$$

Celkový vzorec pro vektorizační metodu TF-IDF je:

$$TF - IDF = TF * IDF$$

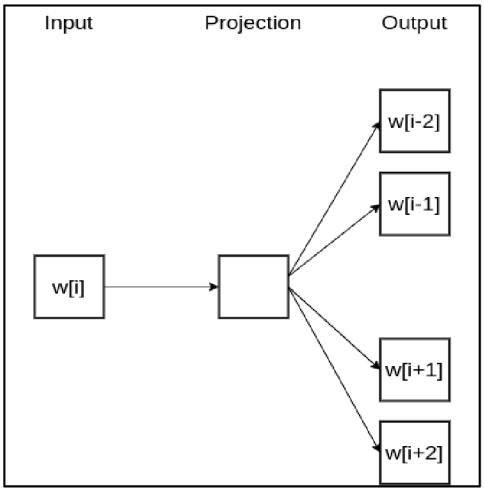
Pro každé slovo v dokumentu tedy vznikne míra jeho důležitosti, což může být také použito v rámci výběru příznaků (feature selection).

3.5.3 Word2Vec

Word2Vec byl zveřejněn v roce 2013 výzkumníky Googlu. Tento přístup využívá sílu jednoduché neuronové sítě ke generování vnořených slov (word embeddings).
V předchozích dvou přístupech bylo s každým slovem zacházeno jako s individuální entitou a sémantika byla zcela ignorována. S uvedením Word2Vec bylo řečeno, že vektorová reprezentace slov má být kontextově vědomá. Protože každé slovo je reprezentováno jako n-rozměrný vektor, lze si představit, že všechna slova jsou mapována do tohoto n-rozměrného prostoru takovým způsobem, že slova mající podobný význam existují v tomto hyperprostoru ve vzájemné těsné blízkosti. Jsou 2 způsoby, jak používat Word2Vec:

- **Skip-Gram** je založen na principu neuronové sítě. Tato metoda poskytuje slovo neuronové síti za účelem předpovědi kontextu.

Obrázek 10 - Neuronová síť skip-gramu



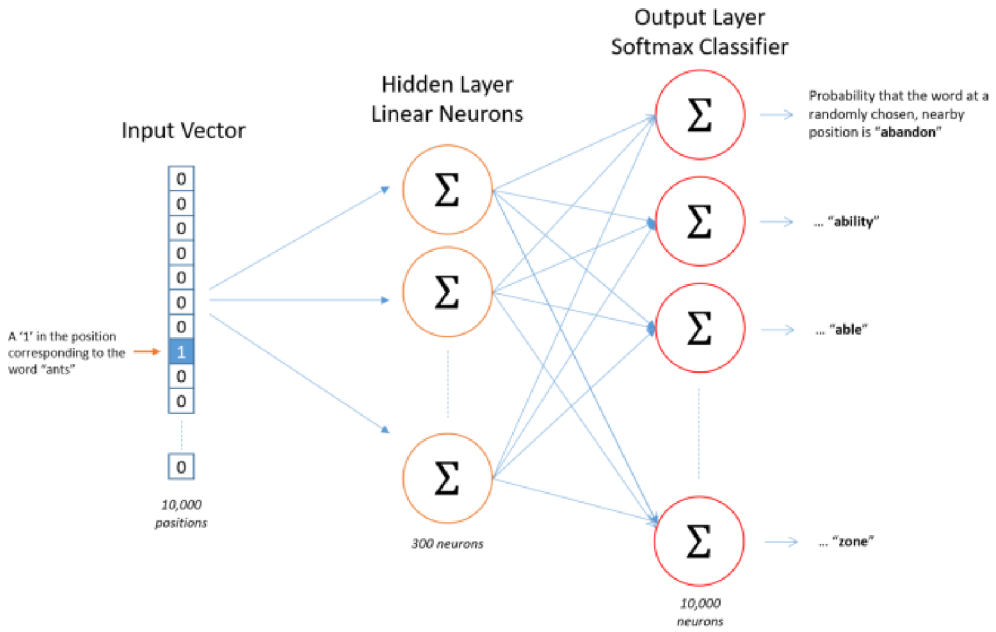
Zdroj: (26)

Na obrázku 10 je w[i] vstupní slovo na „i“ místě ve větě a výstup obsahuje dvě předchozí slova a dvě následující slova vzhledem k „i“. Cílem předpovědi je pravděpodobnost, že slovo bude kontextovým slovem pro dané cílové slovo. Výstupní pravděpodobnosti vycházející ze sítě nám řeknou, jaká je pravděpodobnost, že najdeme každé slovo ze slovní zásoby poblíž našeho vstupního slova. Tato síť se skládá ze vstupní vrstvy, jedné skryté vrstvy a výstupní vrstvy.

Zajímavé však je, že tuto neuronovou síť ve skutečnosti nepoužíváme. Místo toho je cílem se naučit váhy skryté vrstvy a přitom správně předpovídat okolní slova. Tyto váhy představují vnořená slova. Kolik sousedních slov bude síť předpovídat, je určeno parametrem zvaným „velikost okna“ (window size), který se rozprostírá v obou směrech slova, tedy doleva i doprava. (26)

Neuronová síť skip-gramu pro korpus s 10000 slov je znázorněna na obrázku 11. Skrytá vrstva je lineární vrstva, tj. není zde aplikována žádná aktivační funkce a optimalizované váhy této vrstvy se stanou naučenými vnořenými slovy. Výstupem sítě je jeden vektor s 10 000 složkami obsahující pro každé slovo v našem slovníku pravděpodobnost, že je kontextovým slovem pro naše vstupní slovo.

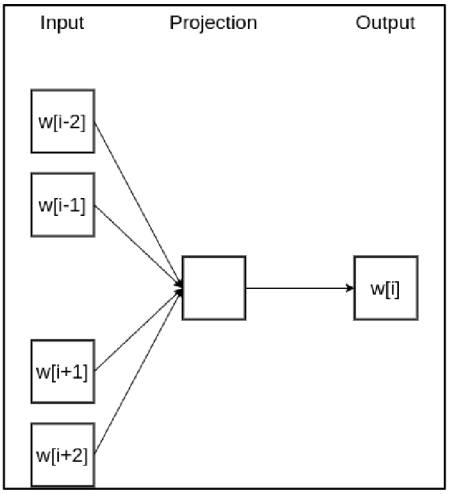
Obrázek 11 - Neuronová síť skip gramu pro 10000 slov



Zdroj: (29)

- **CBOW (Continuous Bag of Words)** je metoda která je zrcadlovým obrazem skip-gramového přístupu. Všechny zápisy zde znamenají přesně to samé jako ve skip-gramu, jen byl přístup obrácen. V přístupu CBOW místo predikce kontextových slov je vložíme na vstup modelu a síť má za úkol předpovědět aktuální slovo, jak lze vidět na obrázku 12.

Obrázek 12- Neuronová síť CBOW



Zdroj: (26)

Cbow se na rozdíl od skip-gramu rychleji natrénuje a lépe reprezentuje frekventovaná slova a naopak skip-gram pracuje lépe s menším korpusem a lépe pracuje se slovy, které se vyskytují málo.

3.5.4 GloVe

GloVe (Global Vectors) byl vyvinut na univerzitě ve Stanfordu. Metoda Word2vec se opírala o parametr „velikosti okna“, který byl vždy ovlivněn pouze okolními slovy v původní větě, což je poněkud neefektivní využití statistiky, protože existuje mnohem více informací, které můžeme získat a pracovat s nimi. GloVe oproti tomu využívá globální i místní statistický přístup k vytvoření vnořených slov.
GloVe používá matici společného výskytu (co-occurence), která říká, jak často se určitá dvojice slov vyskytuje společně. Každá hodnota v matici společného výskytu představuje dvojici slov vyskytujících se společně. Za předpokladu, že korpus má V slov, bude matice společného výskytu X o rozměrech V x V, kde i je řádek a j sloupec. X_ij udává, kolikrát se slovo i vyskytlo společně se slovem j. Příklad matice společného výskytu pro věty *„pes sedí na stole“ a „na stole je jídlo“* s velikostí okna 1 bude vypadat následovně:

Tabulka 1 - Matice společného výskytu pro GloVe

	pes	sedí	na	stole	je	jídlo
pes	0	1	0	0	0	0
sedí	1	0	1	0	0	0
na	0	1	0	2	0	0
stole	0	0	2	0	1	0
je	0	0	0	1	0	1
jídlo	0	0	0	0	1	0

Zdroj: vlastní zpracování

Řádky a sloupce jsou tvořeny slovníkem, který se v tomto případě skládá pouze ze dvou vět. Bylo zjištěno, že GloVe má lepší výsledky oproti ostatním vektorizačním technikám v úlohách jako je podobnost slov a rozpoznávání pojmenovaných entit. GloVe také dobře zachycuje sémantiku slov s malým výskytem a dobře funguje na malé korpusy. (26)

3.5.5 FastText

FastText byl představen Facebookem v roce 2016. Myšlenka FastTextu je velmi podobná Word2Vec metodě. FastText se oproti jiným metodám ovšem zlepšil díky své schopnosti zobecnění na neznámá slova, která u ostatních metod zatím chyběla.

Namísto použití slov k vytváření vnořených slov jde FastText o jednu úroveň hlouběji a to na úroveň písmen. Použití znaků místo slov přináší výhodu zejména pro jazyky, které mají různé tvary slov se stejným významem, jako třeba skloňování podstatných jmen nebo různé tvary sloves v českém jazyce. Další výhoda je v tom, že pro trénování je potřeba méně dat, protože slovo se svým způsobem stává svým vlastním kontextem, což má za následek mnohem více informací, které lze extrahovat z části textu. (26)

3.6 Validační přístupy

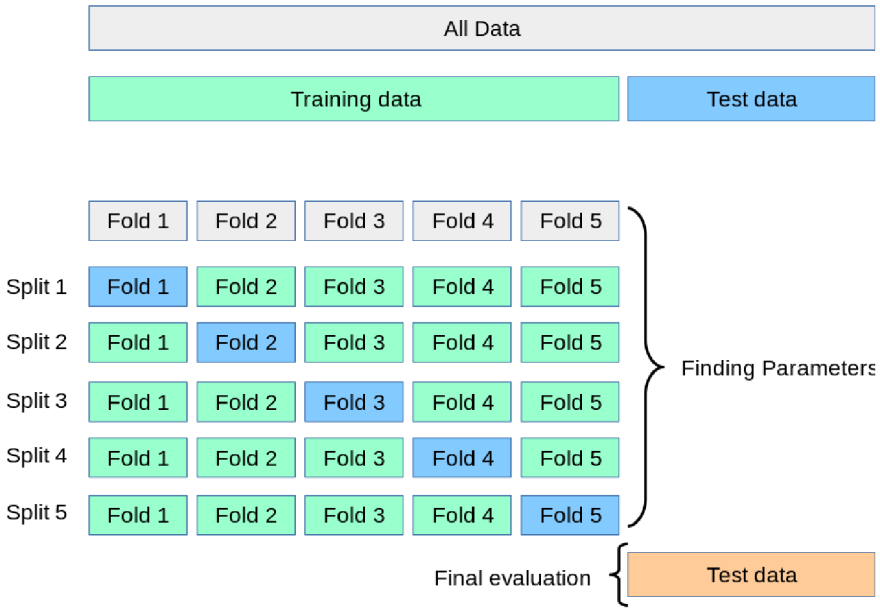
Základem úloh strojového učení s učitelem je rozdělit data na trénovací a testovací sadu. Trénovací sada slouží k natrénování samotného modelu a na testovací sadě se ověřuje výkonnost daného modelu. Vývoj a trénování modelu však může zahrnovat něco, čemu se říká ladění hyperparametrů (hyperparameter tuning), což je proces konfigurace a nastavení daného učebního algoritmu. V případě trénování několika různých nastavení modelu a následné vyhodnocení na testovacích datech může dojít k tzv. úniku informací (information leaks). Tento únik informací může vést k tomu, že model strojového učení bude přeučený (overfitting). Tento stav má za následek, že model nebude schopný dobře generalizovat na nová data, která ještě nikdy neviděl. Proto je vhodné použit rozdělení datové sady na trénovací, validační, na které se ladí hyperparametry a testovací sadu, na které se provede finální ověření. Toto klasické rozdělení, kterému se říká jednoduchá validace má však své nevýhody. Hlavní z nich nastává, pokud je nedostatečně velká datová sada, která po rozdělení na 3 skupiny není statisticky reprezentativní. Proto se používají další validační přístupy. (2, s. 100)

3.6.1 K-násobná křížová validace

K-násobná křížová validace je velice populární přístup rozdělení dat, kde testovací sada držena stranou a validační sada se vytváří z trénovacích dat. V základním přístupu je tréninková sada rozdělena na *k* menších sad. Pro každý z *k* se dodržuje následující postup: Model je trénován za použití *k – 1* dat jako trénovací data a poslední díl je použit na validační data. Tento proces se opakuje *k* krát.

Konvenční přístupy bývají volit *k*=10, případně *k*=5, ale lze zvolit jakékoliv číslo menší, než je počet vzorků. Na obrázku 13 je zobrazena 5-násobná křížová validace. Výsledná výkonost modelu se následně počítá jako průměr všech jednotlivých běhů. (30)

Obrázek 13 - Křížová validace



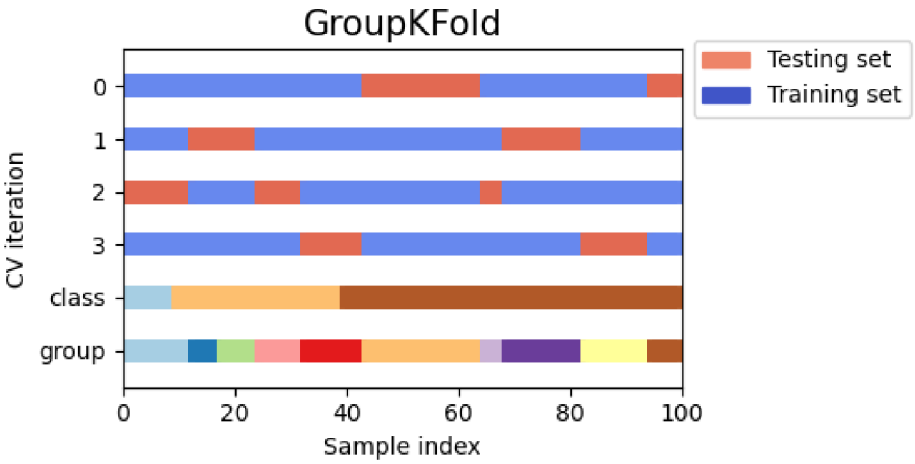
Zdroj: (30)

Mezi další techniky, které mohou být použity při křížové validaci patří stratifikovaný výběr, kdy rozdělení mezi tréninkové a validační data bere v potaz poměr jednotlivých tříd z cílové proměnné. Tento poměr zachovává jak v trénovací tak validační sadě stejný. Tento přístup je vhodný zejména pro nevyvážené datové soubory.

Další technikou je skupinová křížová validace (group k-fold), která zajišťuje, že stejná skupina není zastoupena v testovacích ani validačních sadách. Například pokud jsou data

získána od různých subjektů s několika vzorky na subjekt. Pokud je v tomto případě model dostatečně flexibilní, aby se naučil specifické rysy jednotlivých subjektů, mohl by selhat při zobecnění na nové subjekty. Data ze stejného subjektu nikdy nejsou v trénovací i validační sadě. Jak znázorňuje obrázek 14, tak v tomto případě nemusí mít jednotlivé běhy k-validace stejný poměr trénovacích a validačních (testovacích) dat vzhledem k nerovnováze v datech. (30)

Obrázek 14 - Skupinová křížová validace



Zdroj: (30)

3.6.2 Leave one out validace

Leave one out je jednoduchá křížová validace. Každá učební sada je vytvořena odebráním všech vzorků kromě jednoho, přičemž testovací sada je tvořena právě jedním tímto vzorkem. Pokud jde o vyhodnocení modelu, tento přístup vede k velkému rozptylu. Model se sice trénuje na $n - 1$ všech vzorcích, takže natrénovaný model je v každém kroku velmi podobný, ovšem vyhodnocení na v každém kroku jiném vzorku přináší velký rozptyl. Obecné pravidlo většina autorů uvádí, že by měla být upřednostněna 5- nebo 10-násobná křížová validace před leave one out validací. (30)

3.6.3 Bootstrap

Bootstrap neboli náhodný výběr s vracením je metoda, kdy se náhodně vybírají vzorky z celé trénovací sady s tím, že vzorky mohou být vybrány i vícekrát. Zbylé vzorky, které

nebyly vybrány na trénování jsou přiřazeny do validační sady. Validační sada může být v každém kroku bootstrapu jinak obsáhlá. Každá jednotlivá část bootstrapu je o stejné velikosti, jako byla původní tréninková sada. Vytváření bootstrapu na 3 sady pro 10 vzorků je zobrazeno na obrázku 15. Výsledné hodnocení modelu se obvykle počítá jako průměr jednotlivých bootstrep sad, stejně jako tomu je u křížové validace. Tato metoda je vhodná pro malé datové sady s vyváženým rozložením tříd, protože uměle navyšuje počet vzorků. (31)

Obrázek 15 - Bootstrap validace

Zdroj: (31)

3.7 Metriky vyhodnocení

K posouzení a vyhodnocení natrénovaného modelu je potřeba použít určité metriky, které udávají, jak je model kvalitní a úspěšný. Pro různé druhy úloh se používají různé druhy metrik, stejně jako tomu je při výběru algoritmu pro trénování modelu. Některé běžné metriky pro klasifikaci jsou například úspěšnost (accuracy), přesnost (precision), pokrytí (recall), F1 skóre, ROC křivka a AUC. Pro regresní úlohy to může být R-Squared skóre, Mean Absolute Error (MAE) nebo Root Mean Squared Error (RMSE).

U úloh učení bez učitele metriky vyhodnocení posuzují spíše kvalitu daného klastru, seskupení nebo asociace, nežli úspěšnost daného modelu. Je to dáno tím, že na trénovacích datech nemáme žádnou cílovou proměnnou a proto nemůžeme využít označených dat z minulosti. Proto je velmi důležitá přesná interpretace výsledků a nejen hodnotící metriky.

3.7.1 Matice záměn

Matice záměn (confusion metrix) je grafické znázornění predikovaných hodnot. Používá se pro klasifikační úlohy a nejčastěji je prezentována formou tabulky. Udává vztah mezi správně a špatně klasifikovanými třídami. Matice záměn pro binární klasifikaci do tříd 1 a 0 je znázorněna tabulkou 2.

Tabulka 2 - Matice záměn

		Predikce	
		0	1
Realita	0	TN	FP
	1	FN	TP

Zdroj: (32)

- TN (True negative) je počet správně predikovaných hodnot třídy 0.
- TP (True positive) je počet správně predikovaných hodnot třídy 1.
- FN (False negative) je počet špatně predikovaných hodnot do třídy 0.
- FP (False positive) je počet špatně predikovaných hodnot do třídy 1.

Matice záměn sama o sobě dává představu o celkové kvalitě modelu, ovšem pro přesnější vyhodnocení byly na jejím základě definovány následující metriky:

Úspěšnost

Úspěšnost (accuracy) je nejjednodušší metrika a lze ji definovat jako počet správně klasifikovaných případů dělený celkovým počtem případů. Tato metrika není příliš vhodná na nevyvážené datové soubory. Uvažujme, že máme úlohu zjišťování podvodů v bankovních transakcích, kde je 99% případů oprávněných transakcí a pouze 1% podvodných. Pokud by

model predikoval všechny transakce jako oprávněné, měl by 99% úspěšnost, ovšem v praxi by byl nepoužitelný. (32) Úspěšnost je dána vztahem:

$$Úspěšnost = \frac{TP + TN}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

Přesnost

Přesnost (precision) je počet správně predikovaných pozitivních případů (hodnot třídy 1) z celkového počtu predikovaných pozitivních případů. Je dána vztahem:

$$Přesnost = \frac{TP}{\mathrm{TP} + \mathrm{FP}}$$

Pokrytí

Pokrytí (recall, True Positive rate) udává počet správně identifikovaných pozitivních případů ze všech pozitivních případů. V případě úlohy odhalování podvodných transakcí by toto byla metrika, kterou by bylo vhodno použít a maximalizovat, jelikož je důležité odhalit podvodné transakce i vzhledem k tomu, že jich je pouze malá část.

$$Pokrytí = \frac{TP}{\mathrm{TP} + \mathrm{FN}}$$

F1 Skóre

F1 skóre je harmonický průměr přesnosti a pokrytí. Dává stejnou váhu těmto dvěma metrikám a je užitečná v případě, kdy oba tyto ukazatele dávají smysl. Jak je patrné ze vzorce, je citlivá na situace, kdy je jedna z metrik velmi nízká, popřípadě nulová.

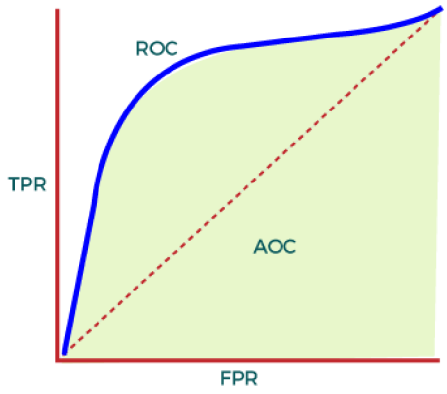
$$F1\ Skóre = 2 * \frac{přesnost * pokrytí}{\mathrm{přesnost} + \mathrm{pokrytí}}$$

3.7.2 Křivka ROC

Křivka ROC (Receiver Operating Characteristic) je graf popisující poměr mezi pokrytím a False Positive rate, kde $FPR = \frac{FP}{\mathrm{FP+TN}}$. Pokud by křivka byla diagonální čárou rozdělující

prostor na dvě poloviny (naznačeno čárkovanou čárou na obr. 16), znamenalo by to, že model má úspěšnost jako náhodně predikující binární klasifikátor.

Obrázek 16 - ROC - AUC křivka



Zdroj: (33)

Oblast pod křivkou ROC se nazývá AUC (Area Under the ROC curve) a vypočítává dvourozměrnou plochu pod celou křivkou ROC v rozsahu od 0 do 1, kde čím vyšší číslo, tím lepší model. AUC se obecně nedoporučuje používat v případě, kdy je velký rozdíl mezi FN and FP. (33)

4 Praktická část práce

Praktická část práce se zabývá řešením klasifikační úlohy na datovém souboru, který se skládá z textových dat vydávaných pojišťovnami a zabývajících se jejich finanční stabilitou a hospodařením. Tento textový dokument se nazývá zpráva o solventnosti a finanční situaci pojišťovny (Solvency and Financial Condition Report, SFCR). Povinnost předkládat SFCR zprávu je stanovená směrnicí 2009/138/EC o přístupu k pojišťovací a zajišťovací činnosti a jejím výkonu, známou též jako Solventnost II. (34)

Předmětem klasifikace je ukazatel, který udává poměr mezi použitelným vlastním kapitálem a solventnostním kapitálovým požadavkem (SCR, Solvency capital requirement) dané pojišťovny. Tento ukazatel se nazývá „Ratio of Eligible own funds to SCR“ a pro tuto úlohu může nabývat dvou hodnot: „stable“ a „positive“. Jedná se tedy o binární klasifikaci. Tento ukazatel je přiřazen na textová i kvantitativní data z předcházejícího roku. Cílem této úlohy je tedy poskytnout klasifikační náhled tohoto ukazatele do následujícího roku.

V praktické části bude aplikováno několik algoritmů strojového učení na datový soubor z hlediska několika variant předzpracování textu. Textové dokumenty jsou psány a budou analyzovány v českém jazyce, což zvyšuje obtížnost procesu předzpracování textu oproti použití Anglického jazyka. Není totiž příliš mnoho zdrojů popisující postup pro předzpracování českého jazyka a zejména chybí podpora Python knihoven a balíčků pro strukturu a sémantiku českého jazyka.

Samotnému modelování bude předcházet proces předzpracování textu pro datový soubor, včetně spojení všech proměnných do jednoho DataFramu. Na vstup modelu kromě očištěného a vektorizovaného textu budou vstupovat také číselné proměnné, které představují indikátory ukazující výkonnostní parametry dané pojišťovny.

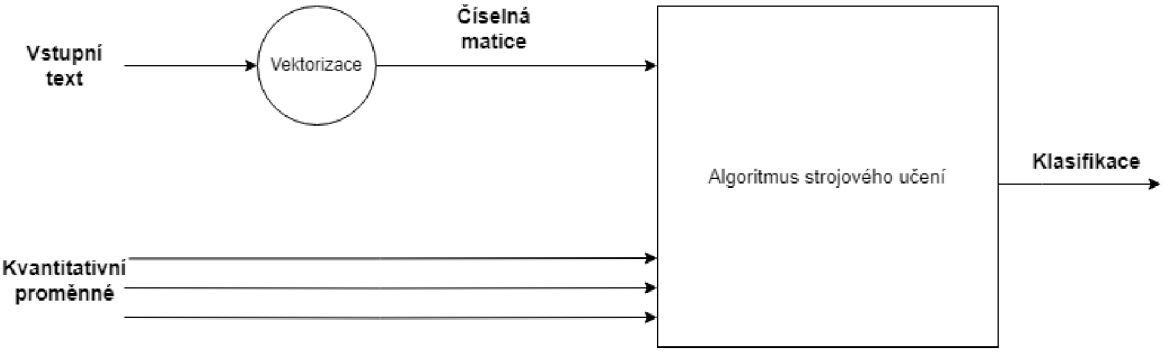
4.1 Datový soubor

SFCR zprávy jsou reportovány ve formátu PDF a informace obsažené v této zprávě jsou zejména popisy finančních rizik, kterým je pojišťovna vystavena, způsoby, jakými pojišťovna hodnotí své aktiva a pasiva a informace o kapitálové situaci pojišťovny. Zpráva rovněž může obsahovat informace o podrobnostech kapitálové struktury pojišťovny,

včetně informací o vlastnictví a řízení pojišťovny. Cílem této zprávy je zajistit transparentnost a poskytnout informace veřejnosti. Zpráva je analyzována regulátorem, ovšem umožňuje každému posoudit rizika spojená s investicemi do dané pojišťovny a důvěryhodnost této společnosti jako zajišťovatele. (35)

Textový korpus je složen ze 70 SFCR dokumentů, které byly nasbírány z let 2019 – 2021. Text z těchto souborů je importován pomocí python skriptu, který je k nalezení v příloze. Kromě textu jako takového jsou součástí těchto dokumentů také QRTs (Quantitative reporting templates) formuláře, které jsou ve většině případů zapsány v tabulkách na konci těchto dokumentů, případně jsou zveřejňovány v samostatných dokumentech. Udávají konkrétní číselné hodnoty o kapitálu, investicích, rizicích a zisku pojišťovny a na základě těchto hodnot jsou vypočítány indikátory, které představují kvantitativní proměnné vstupující do modelu, což je zobrazeno na obrázku 17. Přidáním číselných proměnných do modelu, které souvisejí s cílovou proměnnou může mít pozitivní vliv na přesnost výsledného modelu. (36)

Obrázek 17 - Postup z hlediska vstupních dat



Zdroj: vlastní zpracování

4.1.1 Příprava kvantitativních dat

Kvantitativní data se skládají ze 7 proměnných, které představují jednotlivé indikátory. Tyto indikátory byly vybrány jako ideální kandidáti pro aplikaci modelu a jejich použití je

plošné pro všechny entity. Popis proměnných znázorňuje tabulka 3. Hodnoty na jejichž základě se dané indikátory vypočítají jsou veřejně dostupné nebo přímo odvoditelné z informací, které pojišťovny standardně zveřejňují na svých internetových stránkách.

Tabulka 3 - Popis kvantitativních proměnných

Kvantitativní proměnné	Popis proměnných
Indicator_80	Podíl očekávaných zisků zahrnutých v budoucích pojistných příspěvcích a použitelných vlastních zdrojů
Indicator_72	Podíl podřízených závazků k použitelným vlastním zdrojům
Indicator_74	Podíl použitelných vlastních zdrojů nejvyšší kvality k požadovanému kapitálu
Indicator_109	Podíl nekótovaných/neobchodovatelných akcií na celém portfoliu akcií
Indicator_65	Vliv odloženého daňového závazku na minimální kapitálový požadavek
Indicator_48	Poměr počtu pojistných smluv, které byly zrušeny, ke všem aktivním pojistným smlouvám v portfoliu pojišťovny.
Indicator_118	Poměr expozice (vystavení) dané instituce vůči 5 nejvyšším emitentům
Indicator_30	Poměr krytí zajišťovací společností pro životní a neživotní pojištění

Zdroj: Vlastní zpracování

Hodnoty jednotlivých proměnných byly podrobeny základní průzkumové analýze, aby byla ověřena vhodnost použití na vstup algoritmu strojového učení. Vzhledem k tomu, že se jedná o poměrové ukazatele, tak většina hodnot je v rozsahu od 0 do 1 a u několika výjimek jsou hodnoty lehce vyšší. Proto není nutné tyto hodnoty normalizovat.

Pouze dvě proměnné nemají žádné chybějící hodnoty, proto je třeba zvolit přístup k jejich ošetření. Proměnné, které mají více, než 40 % chybějících hodnot jsou z modelu odstraněny jako celek. Nahrazením chybějících hodnot u takto velké části vzorků by mohlo do modelu vnést určitý šum a negativní vliv. Na základě tohoto kritéria bylo 3 ze 7 vstupních proměnných odebráno. Pro ostatní záznamy, kde je méně, než 40 % chybějících hodnot byly tyto hodnoty nahrazeny průměrnou hodnotou dané proměnné.

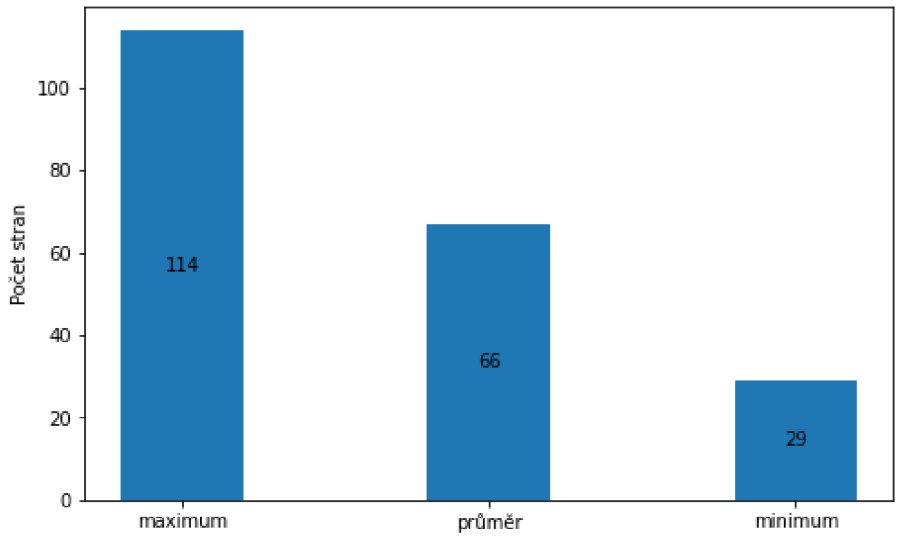
4.1.2 Příprava textových dat

Textový korpus složen ze 70 SFCR dokumentů byl pomocí Python scriptu naimportován do DataFramu. Kód je přiložen v příloze práce. Všechny tyto dokumenty jsou uloženy ve formátu PDF a musejí mít strukturu dle následujících kapitol:

- Kapitola A – Činnost a výsledky
- Kapitola B – Řídící a kontrolní systém
- Kapitola C – Rizikový profil
- Kapitola D – Oceňování pro účely solventnosti
- Kapitole E – Řízení kapitálu

Počet stran dokumentů jednotlivých pojišťoven, stejně jako délka textu v dokumentech, se může lišit. Pro lepší představu je přiložen obrázek 18.

Obrázek 18 - Počet stran SFCR zpráv



Zdroj: Vlastní zpracování

Po naimportování dokumentů následuje čištění textu a tokenizace. Nejprve byla odebrána interpunkce, následně byla jednotlivá slova rozdělena do tokenů. Byla tedy použita slovní tokenizace. Dále z důvodu sjednocení proběhla změna velkých písmen na malé. Také byla

odstraněna slova, případně znaky, které mají méně, než 3 písmena a které v sobě zahrnují čísla. Tyto slova nemají příliš velký sémantický význam z hlediska celého dokumentu. Následovalo odebrání stop slov. Slovník stop slov byl naimportován z textového dokumentu a zahrnuje především spojky a předložky a další často se vyskytující slova, která rovněž nemají velký sémantický význam.

V dalším kroku bylo potřeba vyřešit stemming, případně lemmatizaci, kde dochází k převedení slov na jejich kořen, případně na stejný slovní základ, aby slova stejného významu v jiném pádu, rodu či tvaru obecně byla sjednocena a na vstupu algoritmu strojového učení tak měla stejný význam. K tomuto úkonu, stejně jako k většině předešlým je vhodno použít jednu z python knihoven určených pro zpracování přirozeného jazyka. Mezi nejpoužívanější knihovny patří například: Natural Language Toolkit (NLTK), TextBlob, CoreNLP, Gensim, spaCy, polyglot, PyNLPl, Pattern.

Žádná z těchto, ani dalších knihoven však nepodporuje stemming nebo lemmatizaci pro český jazyk. Proto byl použit přístup, kdy všechna slova byla zkrácena na prvních pět písmen. Tím je u většiny slov odstraněna přípona a koncovka a vznikají tak jednotná slova, která by jinak byla kvůli jiným sufixům brána jako různá.

Další redukce proběhla pro slova, která se i po odebrání sufixů v celém textovém korpusu objevila pouze jednou. Tato slova jsou z velké části URL webových stánek, neobvyklé zkratky, překlepy či chybně naimportovaná slova a proto jsou pro náš model nežádoucí.

Celkový korpus je složen ze všech 70 dokumentů a má následující charakteristiky:

Tabulka 4 - Charakteristiky korpusu

Počet stránek celkem	4620
Počet slov před čištěním textu	1668477
Počet unikátních slov před čištěním textu	41607
Počet slov po čištění textu	1061860
Počet unikátních slov po čištění textu (včetně odebrání sufixu a jednovýskytových slov)	7905

Zdroj: Vlastní zpracování

Na všechny jednotlivé kroky předzpracování textu byly vytvořeny funkce, které byly následně aplikovány na daný text. Po předzpracování textu a spojení textových i kvantitativních proměnných vznikl dataframe, který je naznačen na následujícím obrázku.

Obrázek 19 - Dataframe po spojení proměnných

	Indicator_109	Indicator_65	Indicator_74	Indicator_80	Solvency_ratio_bucket	cleaned_text
0	0.121618	-0.156524	2.318696	0.180849	stable	[obsah, obsah, sezna, tabul, obráz, grafů, přílo, úvodn, slovo, předs, předs, shrnu, vyjád, kapi...
1	0.089989	-0.062071	1.412173	0.208130	stable	[techn, rezer, vykáz, všech, pojis, zajis, závaz, vůči, pojis, opráv, osobá, pojis, zajis, smluv...
2	0.123208	-0.208921	1.828044	0.639396	stable	[rozva, spole, axa, život, pojiš, předp, solve, sesta, prosi, rozva, sesta, soula, naříz, solve,...
3	0.855598	-0.143941	2.128660	0.416517	stable	[bnp, parib, cardi, pojiš, obsah, úvod, rozsa, činno, finan, výsle, rozsa, činno, zákla, údaje, ...
4	0.151241	-0.072778	2.229999	0.663540	positive	[zpráv, solve, finan, situa, obsah, shrnu, zhodn, roku, oceňo, model, alter, metod, oceňo, činno...
...	...	...	...	...	...	...
65	0.193893	-0.072778	2.417105	0.526921	positive	[obsah, souhr, činno, výsle, činno, výsle, oblas, upiso, výsle, oblas, inves, výsle, jinýc, obla...
66	0.686716	-0.072778	9.165577	0.208130	positive	[obsah, úvod, shrnu, činno, výkon, pojiš, činno, pojiš, výsle, pojiš, oblas, upiso, výsle, pojiš,...
67	0.549533	-0.006940	1.107601	0.001184	stable	[obsah, shrnu, činno, výsle, činno, výsle, oblas, upiso, výsle, oblas, inves, výsle, jinýc, obla...
68	0.130602	-0.072778	1.836707	0.465798	stable	[zpráv, solve, finan, situa, obsah, shrnu, zhodn, roku, oceňo, model, alter, metod, oceňo, činno,...
69	0.027828	-0.234568	2.121184	0.614978	positive	[obsah, shrnu, činno, výsle, činno, profi, spole, vznik, akcio, zázem, nabíd, pojiš, činno, poji...

70 rows × 6 columns

Zdroj: Vlastní zpracování

4.1.3 TF-IDF vektorizace

Posledním krokem před samotný trénováním algoritmu strojového učení je vektorizace textu, kdy je potřeba očištěný text převést na matici číselného vektoru. Místo sloupce s očištěným textem vznikne více číselných sloupců, které reprezentují daný text. Pro tuto úlohu byl zvolen TF-IDF algoritmus, který převede každé slovo nebo n-gram v korpusu na proměnnou a přiřadí mu hodnotu dle daného vzorce.

Pro tuto úlohu byly aplikovány 4 různé druhy nastavení vektorizace z pohledu n-gramů. Zvolený rozsah n-gramů a celkový rozsah vstupní matice jsou zobrazeny v tabulce 5. Dataframe po vektorizaci textu pro N-gram rozsah (1,1) je na obrázku 20.

Tabulka 5 - Zvolené n-gramy

N-gram rozsah	Zahrnuté n-gramy	Rozsah tabulky
(1,1)	unigram	70x7910
(1,3)	unigram, bigram, trigram	70x529551
(3,3)	trigram	70x342474
(5,5)	fivegram	70x454103

Zdroj: Vlastní zpracování

Z důvodu velkého rozsahu proměnných je také pro jednotlivé modely aplikována technika výběru příznaků (feature selection), kdy je v trénování modelu zahrnuta jen část proměnných. Tyto proměnné jsou vybrány na základě vlivu na cílovou proměnnou. Většina algoritmů má zabudovaný atribut, který po natrénování na všech proměnných vrátí koeficient důležitosti jednotlivých proměnných. Na základě tohoto koeficientu lze uspořádat proměnné a vybrat jejich určitou část.

Obrázek 20 - Dataframe po vektorizaci unigramů

	aaa	abnor	aborb	abruz	abs	absen	absol	absor	abstr	abych	...	živá	živán	*Kombi*	*PAIDR*	*poměr*	Indicator_80	Indicator_74	Indicator_109	Indicator_65
0	0.002676	0.00181	0.0	0.0	0.000000	0.001909	0.002402	0.002436	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.180849	2.318696	0.121618	-0.156524
1	0.000000	0.00000	0.0	0.0	0.000000	0.001228	0.001236	0.003762	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.208130	1.412173	0.089989	-0.062071
2	0.000000	0.00000	0.0	0.0	0.000000	0.001193	0.001201	0.003653	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.639396	1.828044	0.123208	-0.208921
3	0.000000	0.00000	0.0	0.0	0.000000	0.001723	0.001734	0.003517	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.416517	2.128660	0.855598	-0.143941
4	0.000000	0.00000	0.0	0.0	0.004206	0.000000	0.005286	0.019145	0.0	0.0	...	0.003057	0.0	0.0	0.0	0.0	0.663540	2.229999	0.151241	-0.072778
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
65	0.000000	0.00000	0.0	0.0	0.000000	0.000000	0.005590	0.000810	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.526921	2.417105	0.193893	-0.072778
66	0.000000	0.00000	0.0	0.0	0.000000	0.009291	0.014028	0.009485	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.208130	9.165577	0.686716	-0.072778
67	0.000000	0.00000	0.0	0.0	0.004076	0.000000	0.007318	0.004453	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.001184	1.107601	0.549533	-0.006940
68	0.000000	0.00000	0.0	0.0	0.004061	0.002897	0.005104	0.019225	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.465798	1.836707	0.130602	-0.072778
69	0.000000	0.00000	0.0	0.0	0.000000	0.000000	0.004072	0.004129	0.0	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.614978	2.121184	0.027828	-0.234568

70 rows × 7910 columns

Zdroj: Vlastní zpracování

4.2 Trénování modelů

Trénování jednotlivých modelů včetně importu a předzpracování textu je zpracováno na platformě JupyterLab za použití naimportovaných knihoven. Samotné trénování modelů je provedeno s použitím knihovny Scikit-learn, což je jedna z nejpoužívanějších knihoven na strojové učení v Pythonu.

Jednotlivé modely byly v první fázi natrénovány v základním (defaultním) nastavení. Data byla rozdělena na trénovací a testovací za použití K – násobného validačního přístupu. Nejběžnější přístup rozdělení na trénovací – validační - testovací sadu byl pro tuto úlohu zavrhnut z důvodu nízkého počtu dostupných záznamů. Pro trénování modelů v základním nastavení a bez použití ladění hyperparametrů by přístup K – násobné validace s rozdělením pouze na trénovací a testovací sadu mohl být efektivním.

Pro tuto úlohu bylo tedy zvoleno rozdělení na trénovací a testovací data formou K – násobné validace. Parametr K byl v tomto případě zvolen $K = 7$, kde pro soubor o vzorku 70 dat vychází každý běh trénování na rozdělení 60 trénovacích a 10 testovacích dat. Tento přístup je znázorněn na obrázku 21. Výsledná úspěšnost je brána jako průměrná úspěšnost všech 7 trénovacích iterací.

Obrázek 21 - 7-násobná validace testovacích dat

	Všechna data						
Trénování 1	Test	Train	Train	Train	Train	Train	Train
Trénování 2	Train	Test	Train	Train	Train	Train	Train
Trénování 3	Train	Train	Test	Train	Train	Train	Train
Trénování 4	Train	Train	Train	Test	Train	Train	Train
Trénování 5	Train	Train	Train	Train	Test	Train	Train
Trénování 6	Train	Train	Train	Train	Train	Test	Train
Trénování 7	Train	Train	Train	Train	Train	Train	Test

Zdroj: Vlastní zpracování

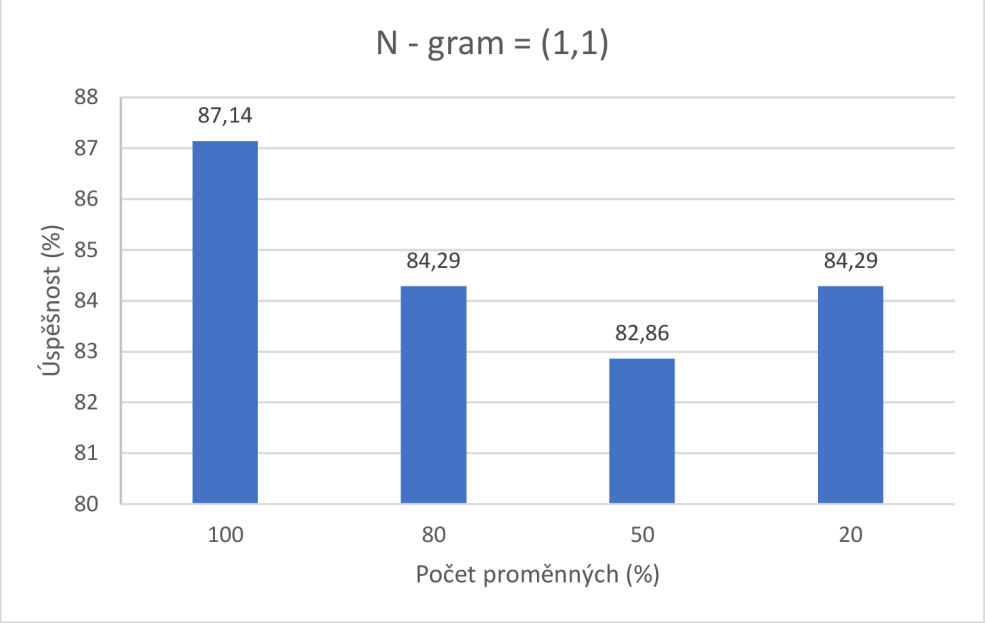
4.2.1 KNN model

Model K-nejbližších sousedů byl trénován s nastaveným parametrem $K = 5$, tedy počítání vzdálenosti pro pět nejbližších sousedů každého bodu. Vzdálenostní metrika byla zvolena parametrem $p = 2$, tedy euklidovská vzdálenost. Největší úspěšnost byla pro vstupní datový soubor, který zahrnuje pouze unigramy a počítá se všemi proměnnými. Úspěšnost klasifikace v tomto nastavení byla 87,14 % a je zobrazena na obrázku 22.

K – nejbližších sousedů je jeden z mála algoritmů, pro které není definován zabudovaný atribut pro počítání důležitosti jednotlivých proměnných. Proto lze zvolit jeden ze

statistických přístupů výběru příznaků, který přímo nesouvisí s daným algoritmem strojvého učení. Mezi takovéto přístupy patří například low variance elimination, univariate feature selection nebo recursive feature elimination. Všechny tyto postupy jsou dostupné v knihovně Scikit-learn. Pro tuto úlohu byl použit univariate feature selection, který využívá ANOVA f-test pro kvantitativní proměnné. (37)

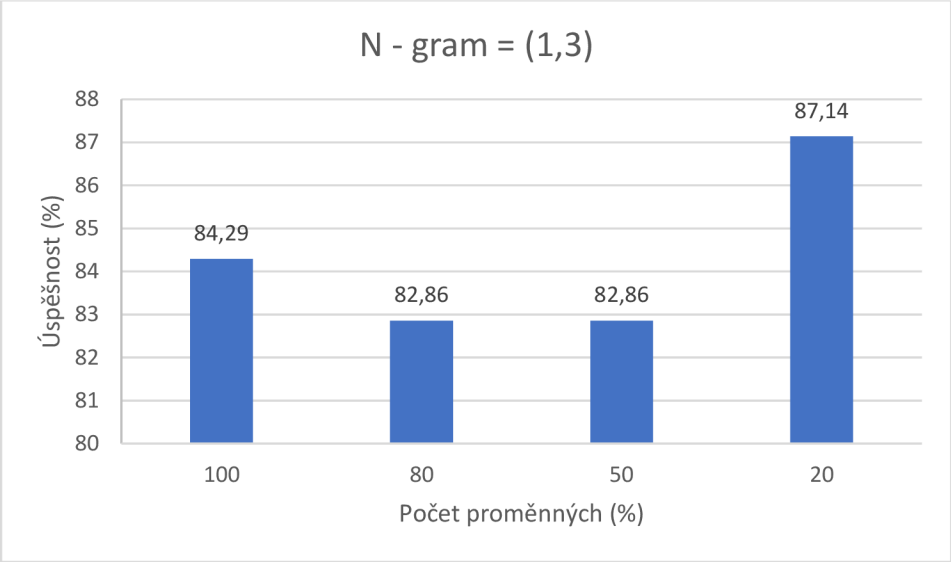
Obrázek 22 - KNN pro N-gram=1



Zdroj: Vlastní zpracování

Podobně dobré výsledky jako pro model s vektorizací unigramů dosahoval model s použitím unigramů, bigramů a trigramů, tedy model s nejvíce proměnnými. Ovšem jak lze na obrázku 23 vidět, pro tento model byla nejlepší úspěšnost s použitím výběru proměnných, kde nejlepších výsledků dosahoval model s 1/5 proměnných. Úspěšnost byla 87,14 %, tedy stejná jako u modelu unigramů se všemi proměnnými.

Obrázek 23 - KNN pro N-gram = (1,3)



Zdroj: Vlastní zpracování

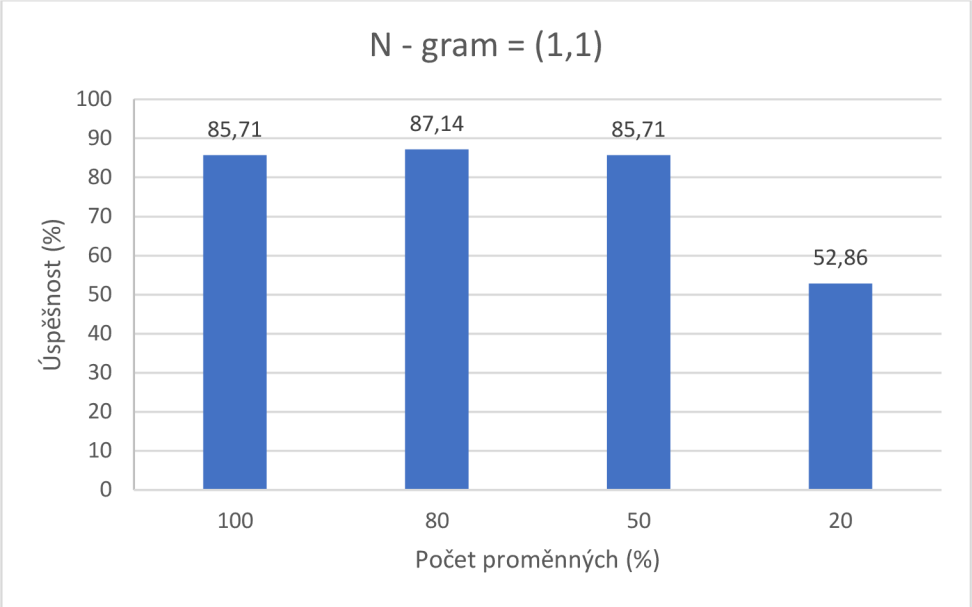
4.2.2 SVM model

Model podpůrných vektorových strojů byl trénován s pomocí modulu sklearn.svm.SVC, který je určen pro klasifikační úlohy s malým nebo středně velkým datovým souborem (co do počtu záznamů, nikoli proměnných). Nabízí využití několika druhů jader, včetně těch, které byly uvedeny v teoretické části práce.

Pro tuto úlohu bylo zvoleno lineární jádro, regularizační parametr byl ponechán v základním nastavením, tedy $C = 1$. Tento parametr určuje trade-off mezi maximalizací mezery mezi nejbližšími body rozdílných třídami a minimalizací chyby klasifikace.

SVM model měl nejvyšší úspěšnost s vektorizací unigramů a 4/5 nejvíce prediktivních proměnných. Úspěšnost v tomto nastavení byla 87,14 %. Úspěšnosti pro trénování unigramů jsou na obrázku 24. Úspěšnosti pro trénování vyššího počtu ngramů byly o něco nižší, kolem 82 % a jsou k nahlédnutí v příloze práce.

Obrázek 24 - SVM pro N-gram = 1



Zdroj: Vlastní zpracování

4.2.3 Model logistické regrese

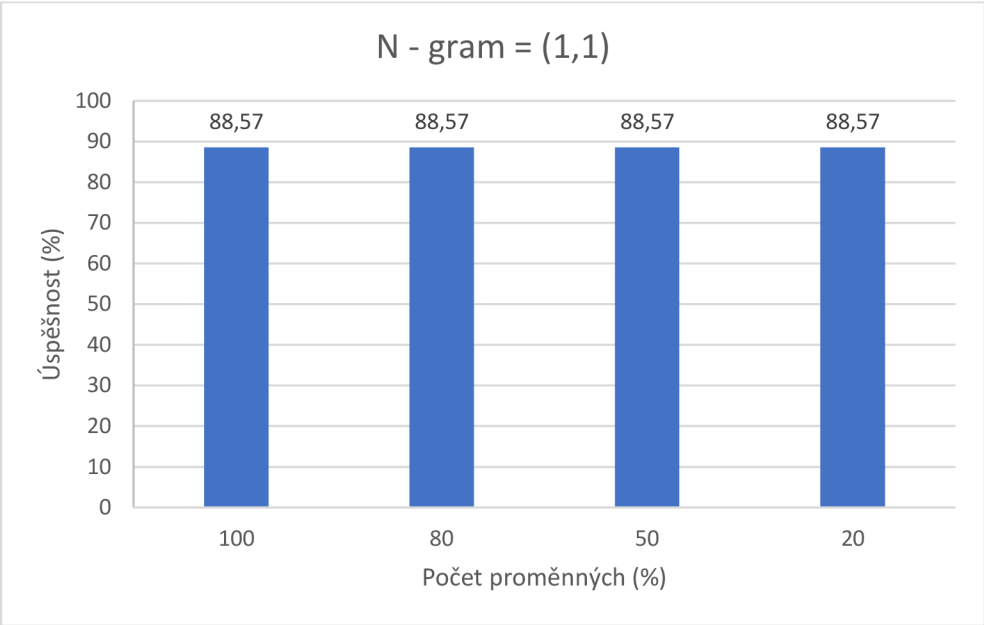
Logistická regrese byla natrénována s regularizačním penalizačním parametrem *l2* neboli ridge regularizací. Při použití této regularizace se do nákladové funkce přidá další člen, který udržuje hodnoty koeficientů modelu co možná nejnižší a model se tím stává jednodušším. Další variantou je *l1* parametr, který představuje lasso regularizaci nebo elasticnet, což je kombonací obou předchozích.

Parametr C byl opět ponechán na hodnotě 1. V logistické regresi tento parametr určuje, jak silně jsou koeficienty modelu penalizovány za velké hodnoty.

Logistická regrese měla největší úspěšnost opět s vektorizací unigramů a to 88,57 %. Byla rovněž provedena selekce proměnných, ovšem na úspěšnost to nemělo žádný vliv. Selekce proměných se v tomto případě vytváří stejně jako u SVM modelu, kdy po natrénování na všech proměnných lze pomocí atributu coef_ získat důležitost každé proměnné a ty méně důležité dle stanovéné procentuální hranice odebrat.

Úspěšnost modelu s vyšším počtem N-gramů se pohybovala mezi 85 % a 87 %, ovšem překvapivě byla lepší úspěšnost s menším počtem zvolených proměnných.

Obrázek 25 - Logistická regrese pro N-gram = 1



Zdroj: Vlastní zpracování

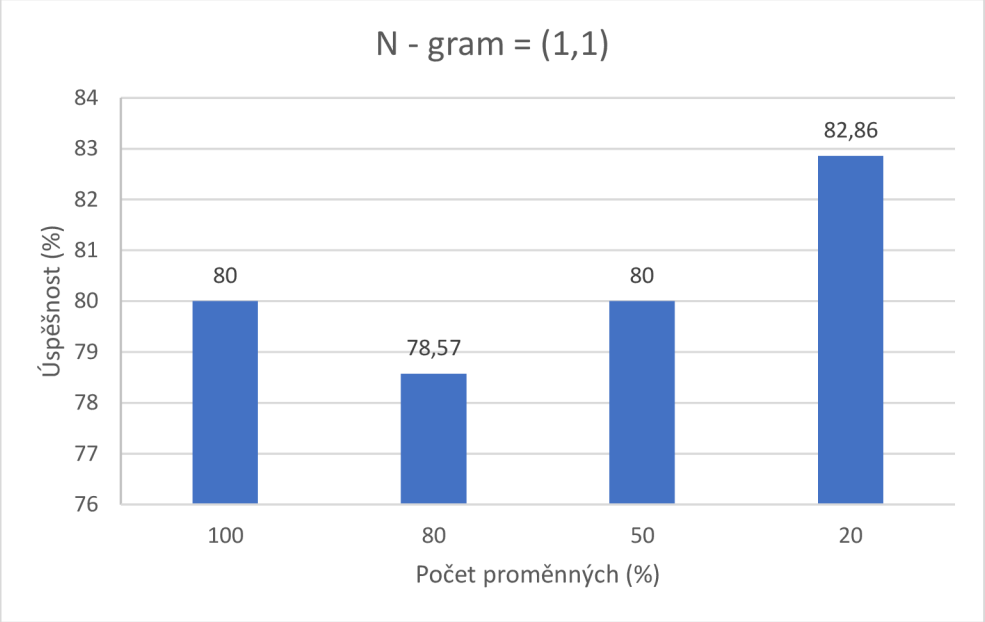
4.2.4 Model rozhodovacího stromu

Pro model rozhodovacího stromu bylo zvoleno jako rozhodovací kritérium Gini index. Scikit-learn pro rozhodovací stromy používá optimalizovanou verzi algoritmu CART. CART v tomto případě konstruuje binární stromy pomocí funkce a prahové hodnoty, které poskytují největší informační zisk v každém uzlu. Hyperparametr max_dept byl ponechán v základním nastavení ‚None'. Tento hyperparametr udává maximální hloubka stromu a určuje jak hluboko, resp. kolik proměnných je na vytvoření jedné větve. Pokud je ‚None', pak se uzly rozšiřují, dokud nejsou všechny listy použity.

Model rozhodovacího stromu měl nejvyšší úspěšnost s použitím unigramů a pro výběr 1/5 nejvíce prediktivních proměnných, jak lze vidět na obrázku 26. Tato úspěšnost byla 82,86 %. Pro ostatní N-gramy byla úspěšnost nižší pouze o 1-2 % s tím, že vyšší úspěšnost byla opět pro 1/5 vybraných proměnných. Grafické zobrazení úspěšnosti pro ostatní N-gramy je přiloženo v příloze práce. V tomto se model rozhodovacího stromu liší oproti ostatním

klasifikátorům, kde nejvyšší úspěšnost je se všemi zahrnutými proměnnými. Ovšem celková úspěšnost je v porovnání s ostatními o něco menší.

Obrázek 26 - Rozhodovací strom pro N-gram = 1



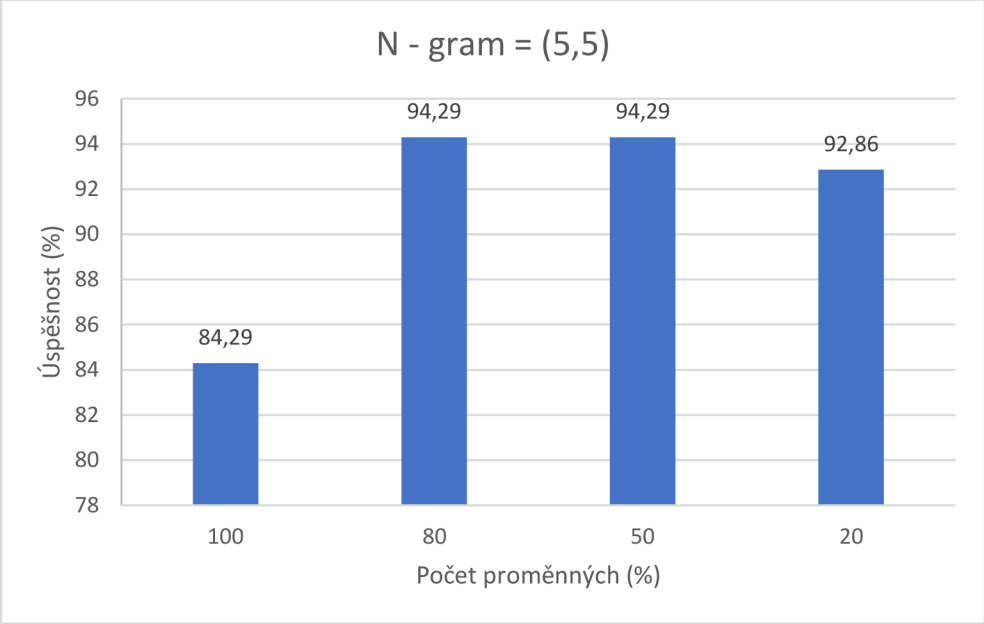
Zdroj: Vlastní zpracování

4.2.5 Naivní Bayesův model

Tento model byl natrénován za použit Gaussova naivního Bayesu, ze třídy sklearn.naive_bayes.GaussiaNB. Jeden z mála hyparametrů pro tento algoritmus je *priors,* který udává apriorní rozdělení pravděpodobnosti pro jednotlivé třídy. V tomto případě byl ponechán na None, což znamená, že apriorní pravděpodobnost je počítána z trénovacích dat.

Naivní Bayes dosahoval vysoké úspěšnosti a to 94,29 % hned v několika případech. V modelu natrénovaném na fivegramech byla tato nejvyšší úspěšnost hned ve dvou případech a to za použití 4/5 a 1/2 proměnných, což znázorňuje obrázek 27. Dále takto vysokou úspěšnost dosahoval model za použití trigramů s 1/5 nejvíce prediktivních proměnných. Tento a další výsledky úspěšnosti pro ostatní modely jsou přiloženy v příloze práce. Obecně lze však konstatovat, že vyšší úspěšnost byla spíše s redukovaným počtem proměnných a že naivní Bayes patří v této úloze k nejlepším klasifikátorům.

Obrázek 27 - Naivní Bayes pro N-gram = (5,5)



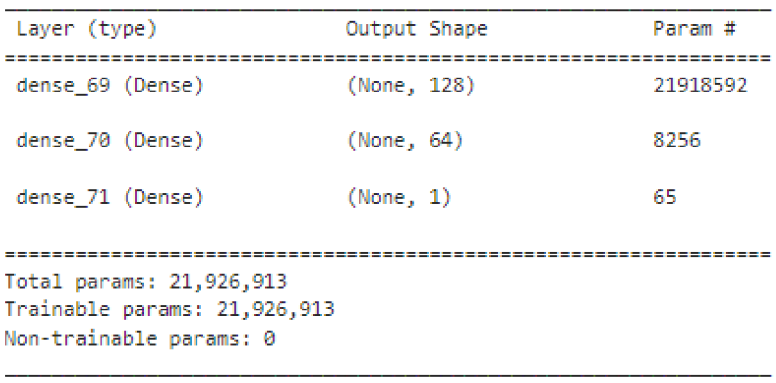
Zdroj: Vlastní zpracování

4.2.6 Model neuronové sítě

K trénování neuronové sítě byly použity knihovny Tensorflow a Keras. Tensorflow je knihovna pro strojové učení a umělou inteligenci. Je populární zejména pro trénování hlubokých neuronových sítí. Knihovna Keras slouží jako rozhraní pro knihovnu Tensorflow.

Model neuronové sítě byl natrénován jako sekvenční neuronová síť, která se skládá ze tří vrstev. Dvě propojené vrstvy s aktivační funkcí ReLU se 128 a 64 neurony a výstupní vrstva s jedním neuronem a sigmoidní aktivací, která zajišťuje výstup binární klasifikace. Nastavení vrstev a počet parametrů pro model s trigramy a 1/2 zvolenými proměnnými je na obrázku 28.

Obrázek 28 - Nastavení modelu pro N-gram (3,3), 1/2 proměnných



Zdroj: Vlastní zpracování

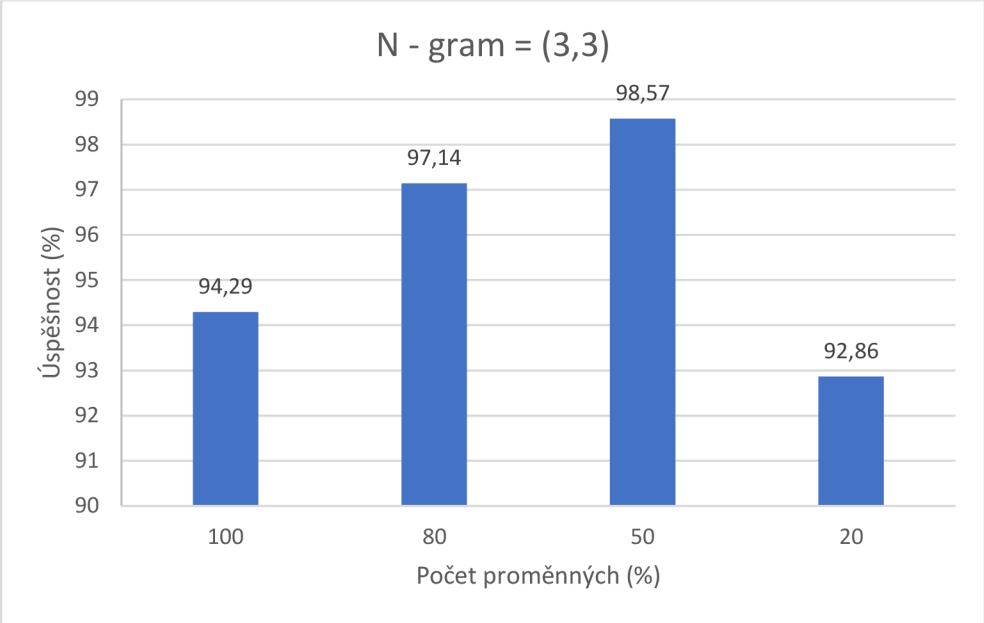
Pro trénování modelu byla použita optimalizační funkce Adam a jako ztrátová funkce byla zvolena binární křížová entropie (binary_crossentropy). Bylo zvoleno 10 epoch trénování, což znamená, že je 10 iterací trénování neuronové sítě, kde v každé z nich prochází data neuronovou sítí a upravují váhy.

Velikost batche (Batch_size) byla zvolena na 128. Tento hyperparametr udává počet trénovacích dat, které jsou použita pro jednu úpravu vah a biasů v rámci jedné epochy. Velikost batche se nastavuje z důvodu, že trénování na celém datasetu může být výpočetně náročné, proto se provádí v tzv. mini-dávkách. Trénování bylo opět provedeno s použitím 7-násobné křížové validace.

Neuronová síť byla trénována jak pro všechny proměnné, tak za použití výběru proměnných metodou univariate feature selection, stejně jako modely KNN nebo naivní Bayes. Rovněž bylo vyzkoušeno různé nastavení n-gramů, jako tomu bylo u předchozích modelů.

Neuronová síť dosahovala velmi vysoké úspěšnosti klasifikace 98,57 % a to ve 3 různých nastaveních pro vstupní data. Pro n-gram rozsah (1,3) to bylo za použití 4/5 proměnných, pro n-gram rozsah (3,3) za použití 1/2 proměnných (obrázek 29) a pro zvolené fivegramy také pro 1/2 použitých proměnných.

Obrázek 29 – Neuronová síť pro N-gram = (3,3)



Zdroj: Vlastní zpracování

Neuronová síť dosahovala nejlepší úspěšnost ze všech zkoušených modelů. Nejhorší výsledky měla v případě modelu unigramů, což je rozdíl oproti předchozím algoritmům, kde unigramy dosahovaly naopak nejlepších výsledků. Výběr proměnných metodou univariate feature selection měl na neuronovu síť také pozitivní vliv, kde nejlepší výsledky byly v případě 80 % a 50 % zahrnutých proměnných.

4.3 Vyladění modelů

Pro zjištění, zdali budou jednotlivé modely náchylné na přeučení a jaké je optimální nastavení parametrů jednotlivých modelů byl zvolen další přístup rozdělení datové sady.

V další části byl datový soubor rozdělen na trénovací a testovací sadu. Trénovací sada byla složena z dat z roku 2019 a 2020, což je 48 záznamů. Testovací sada jsou data z roku 2021 – 22 záznamů. Toto rozdělení nejlépe reflektuje případ, kdy by byl model nasazen do provozu. Vzhledem k malému množství trénovacích dat byl zvolen přístup Leave one out validace, kde vznikne 48 trénovacích běhů a v každém z nich bude model trénován na 47 vzorcích a validován na 1 vzorku. Výsledná validační přesnost je opět dána průměrem jednotlivých běhů.

4.3.1 Ladění hyperparametrů

Pro nalezení optimálních parametrů jednotlivých modelů byla použita technika GridSearch. GridSearch je funkce z knihovny Scikit-learn, která umožňuje systematicky prozkoumat kombinace hyperparametrů pro model a vybrat ty, které vedou k nejlepším výsledkům. Jednotlivé nastavení se nejdříve zapíše do proměnné typu slovník, kde klíč je název hyperparametru a hodnota je výčet hodnot k otestování. Po natrénování na trénovacích a validačních datech je navrácena nejlepší kombinace parametrů a úspěšnost klasifikace dané kombinace. Přehled nastavených parametrů, včetně výsledku GridSearch je v tabulce 6.

Tabulka 6 - GridSearch pro všechny modely

Model	Nastavené parametry pro GridSearch	Výsledné nejlepší nastavení
KNN	{'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 13, 21, 34], 'weights': ['uniform', 'distance'], 'p': [1, 2, 3]}	{'n_neighbors': 3, 'weights': 'distance', 'p': 1}
SVM	{'C': [0.1, 1, 5, 10, 100], 'kernel': ['linear', 'rbf', 'poly', 'sigmoid'], 'gamma': ['scale', 'auto', 0.1, 1, 10]}	{'C': 5, 'kernel': 'linear', 'gamma': 'scale'}
Logistická regrese	{'penalty': ['none', 'l1', 'l2', 'elasticnet'], 'C': [0.1, 1, 2, 3, 4, 10], 'solver': ['lbfgs', 'liblinear']}	{'penalty': 'l2', 'C': 10, 'solver': 'lbfgs'}
Rozhodovací stromy	{'criterion': ['gini', 'entropy', 'log_loss'], 'max_depth': [None, 5, 10, 15, 30], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4, 8], 'max_features': [None, 'sqrt', 'log2']}	{'criterion': 'gini', 'max_depth': None, 'min_samples_split':10, 'min_samples_leaf': 1, 'max_features': None}
Naivní bayes	{'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]}	{'var_smoothing': 1e-07}
Neuronová síť	{'batch_size': [64, 128], 'epochs': [5, 10], 'optimizer': ['adam', 'sgd'], 'activation': ['relu', 'tanh'], 'hidden1': [64, 128], 'hidden2': [64]}	{'batch_size': 64, 'epochs': 5, 'optimizer': 'adam', 'activation': 'relu', 'hidden1': 128, 'hidden2': 64}

Zdroj: Vlastní zpracování

Nastavení parametrů pro K-Nejbližších sousedů je důležité zejména v počtu sousedních bodů, na základě kterých je klasifikován každý bod. Toto udává parametr *n_neighbors*, jehož nejlepší výsledné nastavení bylo rovno 3. Další důležitý parametr byl *weights*, který v defaultní hodnotě *´uniform´* přikládá všem sousedním bodům stejnou váhu, zatímco při nastavení *´distance´* stanovuje váhu sousedních bodů na základě jejich vzdálenosti.

SVM a Logistická regrese mají 3 hyperparametry. Společným z nich je parametr *C*, který udává míru regularizace. Cílem této regularizace je omezit složitost modelu a snížit riziko přeučení. *C* však udává inverzní hodnotu regularizace. Menší hodnota *C* tedy udává silnější regularizaci. *Gamma* koeficient udává koeficient jádra, který ovšem pro lineární jádro nemá žádný vliv, proto je ve výsledném modelu *scale,* což je defaultní nastavení.

Rozhodovací stromy mají mnoho parametrů k nastavení. Jedním z nich je kritérium pro vyhodnocení kvality rozdělení dat při tvorbě stromu. *Criterion: 'gini'* je defaultním nastavením a bylo zároveň i nejlepším nastavením. Další parametry specifikují hloubkou stromu a počet vzorků potřebných k rozdělení rozhodovacího uzlu stromu nebo ke vzniku listu. U rozhodovacího stromu je laděním hyperparametrů velký potenciál ke zlepšení výkonnosti modelu.

Naivní bayes patří mezi nejjednodušší algoritmy, proto je nastaven pouze jeden hyperparametr. Neočekává se tedy velká změna v úspěšnosti modelu oproti předchozímu přístupu.

Neuronové sítě mají mnoho parametrů, které je možno ladit. Kvůli výpočetní složitosti tohoto algoritmu však není možno pokrýt mnoho možností. Tato neuronová síť byla vyladěna opět pro 2 skryté vrstvy. Optimalizační funkce pro nejlepší model byla Adam a aktivační funkce ReLu, stejně jako tomu bylo v defaultním nastavení. Změna byla pouze u parametrů *Batch_size* a *epochs.*

4.3.2 Trénování finálních modelů

Jednotlivé modely byly natrénovány pro odlišné předzpracování textu, dle nejlepších výsledků pro každý model z předchozí části (4.2 Trénování modelů). V tabulce 7 je zaznamenáno pro jaké předzpracování textu byly jednotlivé modely natrénovány.

Tabulka 7 - Finální vyladěné modely

Model	Zvolené n-gramy	Podíl zvolených proměnných	Počet proměnných	Počet kombinací parametrů	Validační úspěšnost	Testovací úspěšnost	Doba zpracování GridSearch
KNN	Unigramy	100%	7909	66	87,50%	90,90%	3 min
SVM	Unigramy	80%	6328	100	87,50%	81,82%	4 min
Logistická regrese	Unigramy	100%	7909	48	87,50%	86,36%	2 min
Rozhodovací stromy	Unigramy	20%	1581	540	87,50%	90,90%	6 min
Naivní Bayes	Fivegram	20%	227054	5	91,67%	86,36%	9 min
Neuronová síť	Trigram	50%	171239	32	93,75%	86,36%	84 min

Zdroj: Vlastní zpracování

Validační úspěšnost byla dána na základě Leave one out křížové validace, jako průměr validačních úspěšností pro jednotlivé běhy. Testovací úspěšnost byla následně vyhodnocena na nových datech pro nejlepší možné nastavení modelů.
V tabulce 7 je rovněž zapsán počet proměnných na vstupu modelu a počet kombinací parametrů při spouštění GridSearch. Tyto dva faktory spolu s náročností daného algoritmu mají vliv na dobu nalezení a natrénování nejvhodnějšího modelu. Z tabulky je zřejmé, že velký vliv na dobu zpracování má počet proměnných, který je pro fivegramy či trigramy až mnohonásobně větší, než unigramy.

Validační úspěšnost prvních 4 modelů byla shodně 87,5 %, pro Naivní Bayes byla o něco lepší a pro Neuronovou síť byla s 93,75 % nejlepší, stejně jako v případě prvotního trénování. Testovací úspěšnost většiny modelů byla velmi podobná té validační. Největší pokles testovací úspěšnosti oproti validační zaznamenaly modely neuronové sítě a SVM. Tento pokles může značit lehké přeučení modelů a neschopnost správné generalizace na nová data. Pro KNN a rozhodovací stromy byla testovací úspěšnost dokonce lehce vyšší, než validační, což je poměrně nezvyklý jev, avšak může být zapříčiněn nedostatečně reprezentativním vzorkem dat pro testování.

5 Výsledky a diskuse

Nejprve byl připraven datový soubor obsahující kvantitativní a textová data, která byla spojena do jednoho DataFramu. Kvantitativní proměnné byly očištěny od chybějících hodnot a text prošel procesem předzpracování, včetně tokenizace. Pro vektorizaci byl zvolen algoritmus TF-IDF, který reprezentuje dokumenty jako vektory a zohledňují výskyt jednotlivých slov v daném dokumentu a jejich relativní důležitost v celém korpusu.

Ačkoliv datový soubor obsahuje pouze 70 dokumentů, tak celkový počet slov v dokumentu je 1.668.477, což po očištění a vektorizaci do unigramů vznikne 7905 unikátních slov, kde každé slovo reprezentuje jednu proměnnou vstupující do modelu. V případě trigramů je to 342469 proměnných a v případě fivegramů dokonce 454098. Tento postup vytváří specifický vstupní datový soubor, který má mnoho proměnných (sloupců), avšak málo záznamů (řádků).

5.1 Srovnání modelů

Srovnání jednotlivých modelů je možné provést z více přístupů. Pro trénování modelů, kde všechny záznamy byly použity na trénování a druhý přístup, kde byla datová sada rozdělena na trénovací a testovací množinu.

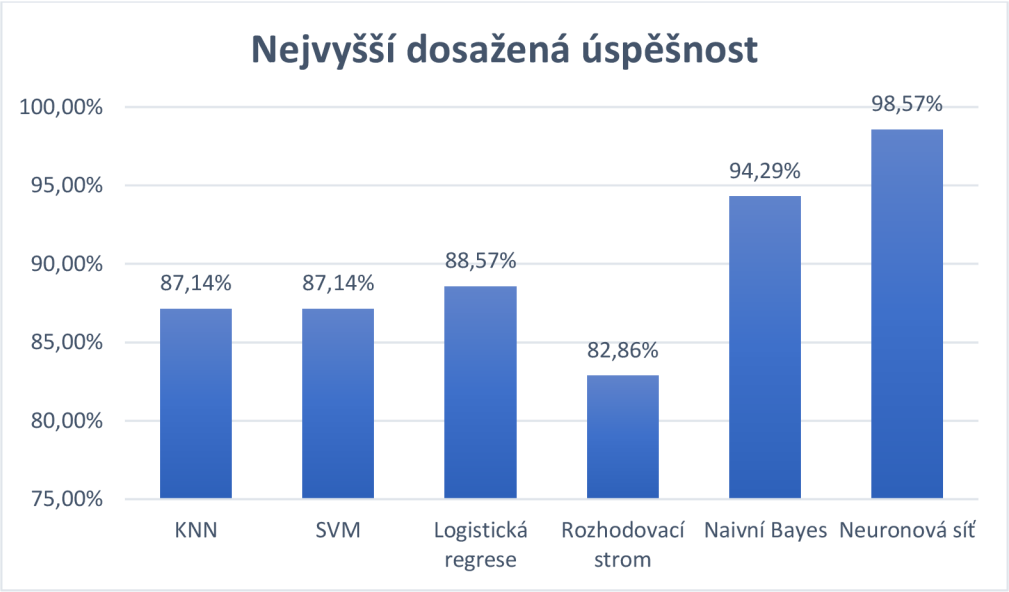
První přístup je zvolen z důvodu nepříliš reprezentativního vzorku dat, takže pro vybrání vhodného modelu je třeba trénovat na všech dostupných datech. Druhý přístup se snaží dodržet konvenční postup pro klasifikační úlohy strojového učení, kde je část datového souboru odložena stranou a použita k finálnímu otestování výkonnosti.

Zvolená metrika pro hodnocení modelů byla úspěšnost. Vzhledem k rovnoměrnému rozložení a stejné důležitosti tříd cílové proměnné nebylo třeba volit jiné metriky vyhodnocení. Bylo vyzkoušeno 6 algoritmů strojového učení pro klasifikaci textu.

5.1.1 Porovnání natrénovaných modelů

V první fázi trénování bylo všech 70 záznamů použito na trénování. Úspěšnost modelu byla měřena 7- násobnou křížovou validací. Na obrázku 30 je vidět nejvyšší dosažená úspěšnost pro každý model.

Obrázek 30 - Porovnání úspěšnosti modelů



Zdroj: Vlastní zpracování

Nejlepší úspěšnost dosahovala neuronová síť a to 98,57 %. Kromě úspěšnosti byla také měřena směrodatná odchylka pro 7 dílčích výsledků křížové validace. Tato směrodatná odchylka udává rozptýlení výsledků napříč 7 běhy křížové validace. Vyšší hodnota směrodatné odchylky naznačuje, že výsledky klasifikace jsou méně konzistentní a mají větší rozptyl. Vyšší hodnota směrodatné odchylky může rovněž indikovat přeučení modelu. Směrodatná odchylka pro neuronovou síť byla 3,5 procentního bodu kolem průměrné hodnoty úspěšnosti jednotlivých běhů.

Druhou nejvyšší úspěšnost měl naivní Bayes s 94,29 %. Směrodatná odchylka pro validační úspěšnosti byla v tomto případě 7,28 %, což značí větší rozptyl validačních úspěšností. Naivní Bayes je na rozdíl od neuronové sítě považován ze jednoduchý algoritmus založený na pravděpodobnostním rozdělení. I přes to celková úspěšnost tohoto klasifikátoru byla velmi dobrá.

Logistická regrese měla rovněž slušnou úspěšnost 88,57 % se směrodatnou odchylkou 6,39 %. Nejlepších výsledků bylo dosaženo pro model unigramů a selekce proměnných neměla na úspěšnost velký vliv, což je rozdíl oproti modelu naivního Bayesu, kde lepší úspěšnost byla pro trigramy a fivegramy a selekce proměnných měla pozitivní vliv na úspěšnost.

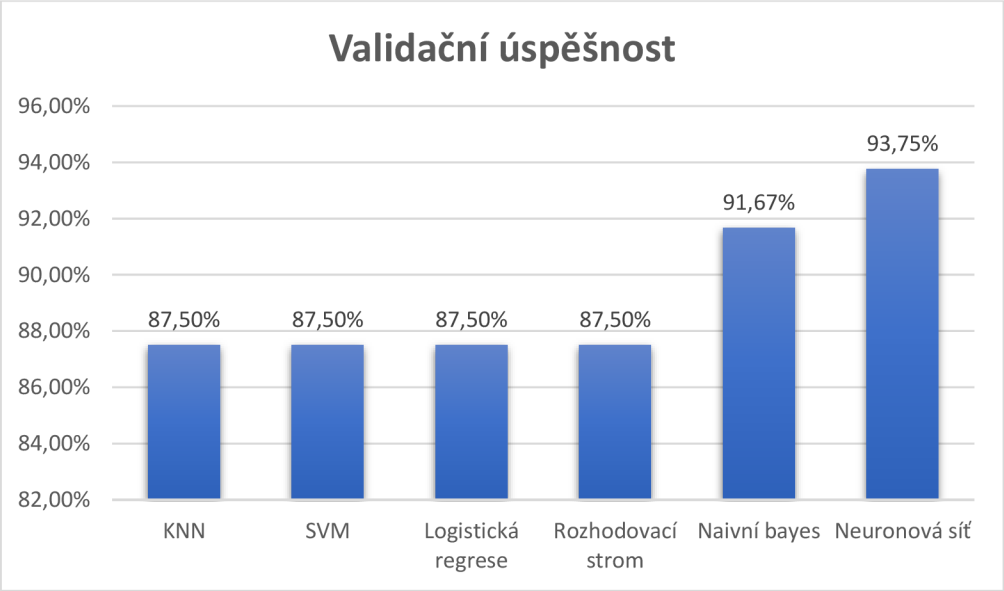
KNN a SVM modely měly shodně úspěšnost 87,14 % s malou směrodatnou odchylkou validačních úspěšností (4,52%). Pro oba algoritmy rovněž vycházel o něco lépe model s unigramy.

Nejhorší úspěšnost 82,86 % vykazoval model rozhodovacího stromu, kde však lze předpokládat zlepšení úspěšnosti pro následný vyladěný model i vzhledem k tomu, že nejlepších úspěšností dosahoval vždy se zvolením pouhých 1/5 nejvíce prediktivních proměnných. Proto další omezení v použití proměnných nebo rozvětvení stromu může zlepšit úspěšnost predikce

5.1.2 Porovnání vyladěných modelů

Vyladění modelů bylo provedeno metodou GridSearch na trénovacích datech obsahujících 48 záznamů. Úspěšnost byla vyhodnocena Leave one out křížovou validací a vyhodnocena jako průměr jednotlivých běhů této validace. Obrázek 31 znázorňuje validační úspěšnosti jednotlivých modelů.

Obrázek 31 - Validační úspěšnost GridSearch



Zdroj: Vlastní zpracování

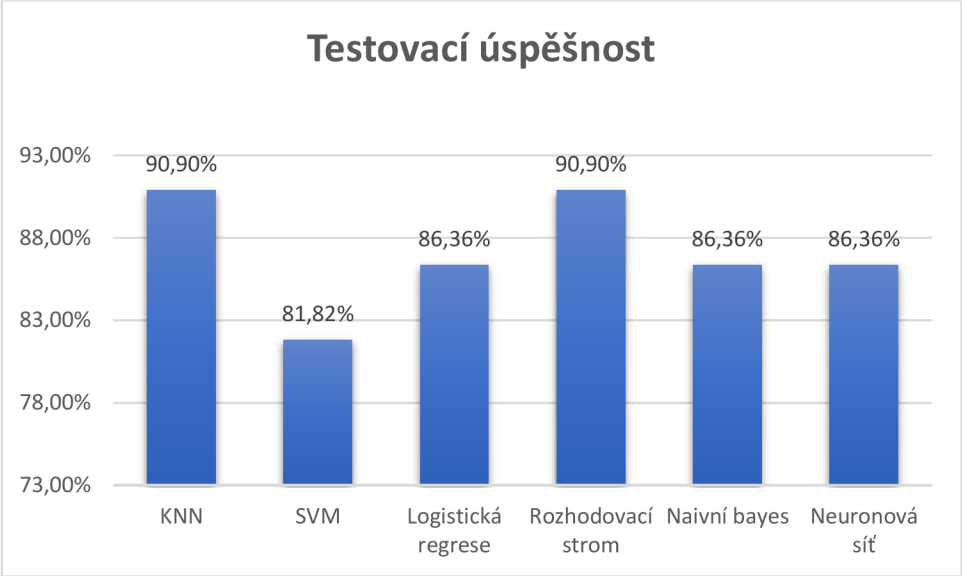
Z obrázku 31 je patrné, že nejvyšší úspěšnost dosahoval stále model neuronové sítě. Ovšem v porovnání s prvotně natrénovaným modelem úspěšnost klesla z 98,57 % na 93,57 % i za předpokladu, že druhý model byl vyladěn na nejlepší možnou kombinaci hyperparametrů. To je dáno tím, že první model byl trénován na všech datech, zatímco druhý pouze na trénovacích a část dat byla ponechána stranou na finální otestování. Z toho lze soudit, že pro takto komplexní algoritmus, jakým je neuronová síť je vhodné použít co největší množství dat na trénování. Mírné zhoršení o 2,6 %, resp. 1 % bylo také u modelu naivního Bayese a logistické regrese.

Pro modely KNN a SVM byla úspěšnost mezi prvním a druhým přístupem téměř stejná. Výrazné zlepšení však bylo u algoritmu rozhodovacího stromu, kdy se použitím GridSearch úspěšnost zlepšila o téměř 5 % i vzhledem k tomu, že část dat byla odložena na testování a na trénování bylo k dispozici menší množství dat.

Jeff Prosis (4) uvádí, že naivní Bayes je vhodným algoritmem pro klasifikaci textu. Toto tvrzení se potvrdilo i na této úloze. Model naivního Bayesu měl po neuronové síti druhou nejvyšší úspěšnost. V případě použití GridSearch algoritmu to bylo 91,67 %.

Finální otestování úspěšnosti proběhlo na testovacích datech, které byly před trénováním modelů odebrány ze vstupního datového souboru. Testovací soubor obsahuje data nasbíraná z roku 2021, zatímco trénovací data byla z předchozích let. Testovací úspěšnost je znázorněna na obrázku 32. Tento přístup byl zvolen z důvodu co nejpřesnějšího nasimulování reálné situace v případě nasazení modelu do praxe. Alternativním a běžně používaným přístupem by bylo data před rozdělením promíchat a následně je náhodně rozdělit na trénovací a testovací.

Obrázek 32 - Testovací úspěšnost vyladěných modelů



Zdroj: Vlastní zpracování

Testovací úspěšnost se u většiny modelů příliš nelišila. Největší pokles oproti trénovací úspěšnosti byl u modelů neuronové sítě a SVM a to méně než 8 %. Tyto modely tedy generalizují o něco hůře, než ostatní modely. Modely rozhodovacího stromu a KNN měly dokonce testovací úspěšnost o 3,4 % vyšší, než validační. Tato zjištění mohou však být velmi ovlivněna malým vzorkem dat v testovacím souboru. Proto by pro lepší důvěryhodnost bylo vhodné nasbírat větší počet dat, případně zkusit jiné rozdělení trénovací a testovací množiny a porovnat, zdali úspěšnosti jednotlivých modelů mají podobné tendence.

Žádný ze zkoušených algoritmů se neprokázal být nepoužitelným. Všechny modely dosahovaly úspěšnosti větší než 80 %. V první fázi byl model rozhodovacího stromu nejhorším, ovšem po vyladění hyperparametrů se jeho validační úspěšnost podstatně zlepšila a testovací úspěšnost měl dokonce nejlepší ze všech modelů spolu s modelem K-nejbližších sousedů. Prosis (4) tvrdí, že rozhodovací stromy mají tendenci podléhat přeučení, což se na této úloze nepotvrdilo. V prvotně natrénovaných modelech byl ovšem nejúspěšnější modelem neuronová síť a následně model naivního Bayesu.

5.2 Nedostatky řešení

Mezi hlavní nedostatky této úlohy patří zejména omezený počet SFCR zpráv, které jsou předmětem klasifikační úlohy. Bylo použito 70 zpráv z let 2019 – 2021, z nichž každá tvoří jeden vzorek. Tento vzorek může zkreslovat vyhodnocení úspěšnosti modelů. Vzhledem k obsáhlosti těchto zpráv bylo však možné získat rozsáhlý textový korpus obsahující 41607 unikátních slov před čištěním textu, což umožňuje natrénování klasifikačních modelů.

Další prostor pro zlepšení je v procesu importování textu z PDF souborů, kdy některé části mohou být naimportovány nesprávně. Chyba v importu je zejména u dokumentů, které jsou ve formátu 2 stránek na jednom listu. Modul Pdfplumber, který je používán za účelem extrakce textu, nedokáže správně načíst tento způsob textu a načítá text po řádcích, ačkoliv se jedná o dvě stránky. Tato chyba nemá vliv pro modely unigramů, které neberou v úvahu pořadí slov ani vět. Pro trigramy a fivegramy to však může mít negativní dopad.

Dalším nedostatkem je chybějící dostupnost knihoven pro předzpracování textu v českém jazyce. Zejména proces lematizace slov by mohl výrazně zlepšit kvalitu vstupního vektoru slov.

Jednou z dalších možností pro zlepšení kvality modelu by bylo vyzkoušet jinou metodu vektorizace textu. V případě většího textového korpusu by bylo možné použít jednu z metod založených na neuronových sítích, jako například Word2Vec.

6 Závěr

Cílem diplomové práce bylo sestavit a porovnat modely strojového učení pro klasifikaci textu. Pro porovnání bylo zvoleno 6 algoritmů strojového učení. Byly vyzkoušeny základní algoritmy, jako jsou K-nejbližších sousedů nebo naivní Bayes, ale také složitější algoritmy, kterými jsou například neuronové sítě, spadající do metod hlubokého učení.

V úlohách zpracování přirozeného jazyka má velkou důležitost proces přípravy a předzpracování textu. Tomuto tématu byla věnována značná část teoretické i praktické části práce. V samotném zpracování úlohy zabrala příprava a předzpracování dat největší časový podíl.

Čištění textu českého jazyka se ukázalo být problematickým z důvodu chybějící podpory českého jazyka u všech běžně používaných knihoven zabývajících se zpracováním přirozeného jazyka. V teoretické části byly také představeny metody vektorizace textu. Pro praktickou část byla vybrána metoda TF-IDF, která bere v potaz četnost a důležitost jednotlivých slov v korpusu.

Datový soubor byl složen ze 70 dokumentů o průměrné délce 66 stran. Každý dokument byl rozřazen do jedné ze tříd „stable“ nebo „positive“ a hodnotí solventnost a finanční situaci pojišťovny z roku následujícího danému dokumentu.

Všechny klasifikační algoritmy se ukázaly být použitelné, s více než 80 % úspěšností. Nejlepší úspěšnosti dosahoval model neuronové sítě, který byl natrénován na všech datech a obsahoval dvě skryté vrstvy. Tento model správně klasifikoval s úspěšností 98,57 %. Dobře klasifikující a zároveň jednoduchý model byl model naivního Bayesu, který dosahoval úspěšnosti 94,29 % a v souladu s literaturou se ukázal být vhodným klasifikačním algoritmem na textová data.

Velkou výzvou pro tuto úlohu je vyřešit problém s menším množstvím dostupných dat. Z tohoto důvodu byly zvoleny méně tradiční způsoby pro rozdělení datového souboru na trénovací a testovací množinu. Ukázalo se, že úspěšnost klasifikace jednotlivých modelů byla velmi dobrá, avšak takto zavedené modely můžou být nevhodné na generalizaci na

nových datech. Proto nedostatečně reprezentativní vzorek dat je jednou z hlavních překážek pro uvedení modelu do provoz.

Možnost dalšího rozšíření úlohy by mohlo spočívat ve změně klasifikačního problému z binární klasifikace na klasifikaci do třech tříd. V případě většího vzorku dat, zejména v případě výskytu pojišťoven v potencionální kategorii „watch“ cílového ukazatele, by bylo vhodné přidat tuto kategorii a vytvořit model pro multiclass klasifikaci.

V době psaní závěru práce (Březen, 2023) vydává OpenAI novou verzi velkého jazykového modelu GPT-4, který má být ještě výkonnější a efektivnější, než jeho předchůdce GPT-3. Tuto událost lze označit jako další krok vpřed v oblasti umělé inteligence a zpracování přirozeného jazyka, kde je vývoj v posledních letech větší než kdy dříve, což může být další motivací pro to, věnovat se tomuto tématu.

7 Seznam použitých zdrojů

(1) What is AI: Oracle. In: *Oracle: What is AI* [online]. 2022 [cit. 2022-11-19]. Dostupné z: https://www.oracle.com/artificial-intelligence/what-is-ai/

(2) CHOLLET, Francois. *Deep learning v jazyku Python: Knihovny Keras, TensorFlow*. 1. Praha: Grada, 2019. ISBN 978-80-247-3100-1.

(3) Co je to strojové učení?. In: *SAP* [online]. [cit. 2023-01-23]. Dostupné z: https://www.sap.com/cz/insights/what-is-machine-learning.html

(4) PROSISE, Jeff. *Applied Machine Learning and AI for Engineers*. 1005 Gravenstein Highway North, Sebastopol, CA 95472.: O'Reilly Media, Inc., 2022. ISBN 978-1-492-09805-8.

(5) GÉRON, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Second edition. Beijing: O'Reilly, 2019. ISBN 978-1-492-03264-9.

(6) VAN HASSELT, Hado. Introduction to Reinforcement Learning. In: *Youtube - DeepMind kanál* [online]. [cit. 2022-12-31]. Dostupné z: https://www.youtube.com/watch?v=TCCjZe0y4Qc

(7) *K-Nearest Neighbor(KNN) Algorithm for Machine Learning* [online]. In: . [cit. 2023-01-01]. Dostupné z: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

(8) *Euclidean_distance* [online]. In: . [cit. 2023-01-01]. Dostupné z: https://wikijii.com/wiki/Euclidean_distance

(9) K-Nearest Neighbors Algorithm. In: *Ibm* [online]. [cit. 2023-01-01]. Dostupné z: https://www.ibm.com/topics/knn

(10) Sklearn.preprocessing.PolynomialFeatures. In: *Scikit-learn.org* [online]. [cit. 2023-01-04]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

(11) SVM Kernels. In: *Dataaspirant* [online]. 2020 [cit. 2023-01-04]. Dostupné z: https://dataaspirant.com/svm-kernels/

(12) Svm multiclass classification. In: *Baeldung* [online]. 2022 [cit. 2023-01-04]. Dostupné z: https://www.baeldung.com/cs/svm-multiclass-classification

(13) Logistic Regression in Machine Learning. In: *Javatpoint* [online]. [cit. 2023-01-05]. Dostupné z: https://www.javatpoint.com/logistic-regression-in-machine-learning

(14) Machine learning decision tree classification algorithm. In: *Javatpoint* [online]. [cit. 2023-01-06]. Dostupné z: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

(15) CHAUHAN, Nagesh Singh. Decision Tree Algorithm, Explained. In: *Kdnuggets* [online]. 2022 [cit. 2023-01-07]. Dostupné z: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html

(16) RAMZAI, Juhi. *Simple guide for Top 2 types of Decision Trees: CHAID & CART* [online]. In: . 2020 [cit. 2023-01-07]. Dostupné z: https://towardsdatascience.com/clearly-explained-top-2-types-of-decision-trees-chaid-cart-8695e441e73e

(17) Naïve Bayes Classifier Algorithm. In: *Javatpoint* [online]. [cit. 2023-03-04]. Dostupné z: https://www.javatpoint.com/machine-learning-naive-bayes-classifier

(18) Naive Bayes. In: *Scikit-learn* [online]. c2007–2023 [cit. 2023-03-04]. Dostupné z: https://scikit-learn.org/stable/modules/naive_bayes.html

(19) Activation functions in Neural Networks. In: *Geeksforgeeks* [online]. 2023 [cit. 2023-03-06]. Dostupné z: https://www.geeksforgeeks.org/activation-functions-neural-networks/

(20) NARASIMMAN, Preethiga. 9 Types of Neural Networks: Applications, Pros, and Cons. In: *Knowledgehut* [online]. 2023 [cit. 2023-03-07]. Dostupné z: https://www.knowledgehut.com/blog/data-science/types-of-neural-networks

(21) BURCHFIEL, ANNI. What is NLP (Natural Language Processing) Tokenization?. In: *Tokenex* [online]. 2022 [cit. 2023-01-08]. Dostupné z: https://www.tokenex.com/blog/ab-what-is-nlp-natural-language-processing-tokenization/

(22) PAI, Aravindpai. What-is-tokenization-nlp. In: *Www.analyticsvidhya.com/blog* [online]. 2020 [cit. 2023-01-09]. Dostupné z: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

(23) Cleaning Text Data. In: *Hex.tech* [online]. [cit. 2023-01-10]. Dostupné z: https://learn.hex.tech/tutorials/concepts/text-data

(24) Stemming vs. Lemmatization in NLP. In: *Towardsdatascience* [online]. 2022 [cit. 2023-01-10]. Dostupné z: https://towardsdatascience.com/stemming-vs-lemmatization-in-nlp-dea008600a0

(25) *Snowball stem* [online]. [cit. 2023-01-10]. Dostupné z: https://snowballstem.org/

(26) JHA, Abhishek. Vectorization Techniques in NLP. In: *Neptune.ai* [online]. 2023 [cit. 2023-01-11]. Dostupné z: https://neptune.ai/blog/vectorization-techniques-in-nlp-guide

(27) GOYAL, Chirag. Word Embedding and Text Vectorization. In: *Analytics Vidhya* [online]. 2021 [cit. 2023-01-12]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/06/part-5-step-by-step-guide-to-master-nlp-text-vectorization-approaches/#h2_7

(28) JOULIN, Armand, Edouard GRAVE, Piotr BOJANOWSKI a Tomas MIKOLOV. Bag of Tricks for Efficient Text Classification. In: *Fasttext* [online]. 2016 [cit. 2023-01-13]. Dostupné z: https://arxiv.org/pdf/1607.01759.pdf

(29) CHABLANI, Manish. Word2Vec (skip-gram model): PART 1. In: *Towardsdatascience* [online]. 2017 [cit. 2023-01-12]. Dostupné z: https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

(30) Cross-validation: evaluating estimator performance. In: *Scikit-learn* [online]. [cit. 2023-03-06]. Dostupné z: https://scikit-learn.org/stable/modules/cross_validation.html

(31) RASCHKA, Sebastian. Model evaluation, model selection, and algorithm selection in machine learning: Part II - Bootstrapping and uncertainties. In: *Sebastianraschka* [online]. 2016 [cit. 2023-03-05]. Dostupné z: https://sebastianraschka.com/blog/2016/model-evaluation-selection-part2.html

(32) GHOSH, Samadrita. The Ultimate Guide to Evaluation and Selection of Models in Machine Learning. In: *Neptune.ai* [online]. 2022 [cit. 2023-01-25]. Dostupné z: https://neptune.ai/blog/the-ultimate-guide-to-evaluation-and-selection-of-models-in-machine-learning

(33) AUC-ROC Curve in Machine Learning. In: *Javatpoint* [online]. [cit. 2023-01-27]. Dostupné z: https://www.javatpoint.com/auc-roc-curve-in-machine-learning

(34) Směrnice 2009/138/ES o přístupu k pojišťovací a zajišťovací činnosti a jejím výkonu (Solventnost II). In: *Eur-Lex* [online]. 2015 [cit. 2023-03-13]. Dostupné z: https://eur-lex.europa.eu/legal-content/CS/LSU/?uri=celex:32009L0138

(35) *Solvency II: Solvency and Financial Condition Report: EIOPA's Supervisory Statement*. 1. Frankfurt: EIOPA, 2017. Dostupné také z: https://www.eiopa.europa.eu/sites/default/files/publications/supervisory_statements/17-310-sfcr-supervisory-statement.pdf

(36) BENGFORT, Benjamin, Rebecca BILBRO a Tony OJEDA. *Applied Text Analysis with Python: Enabling Language-Aware Data Products with Machine Learning*. First Release. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, 2018. ISBN 9781491963043.

(37) Feature selection. In: *Scikit-learn* [online]. [cit. 2023-03-01]. Dostupné z: https://scikit-learn.org/stable/modules/feature_selection.html

8 Seznam obrázků, tabulek a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam použitých zkratek

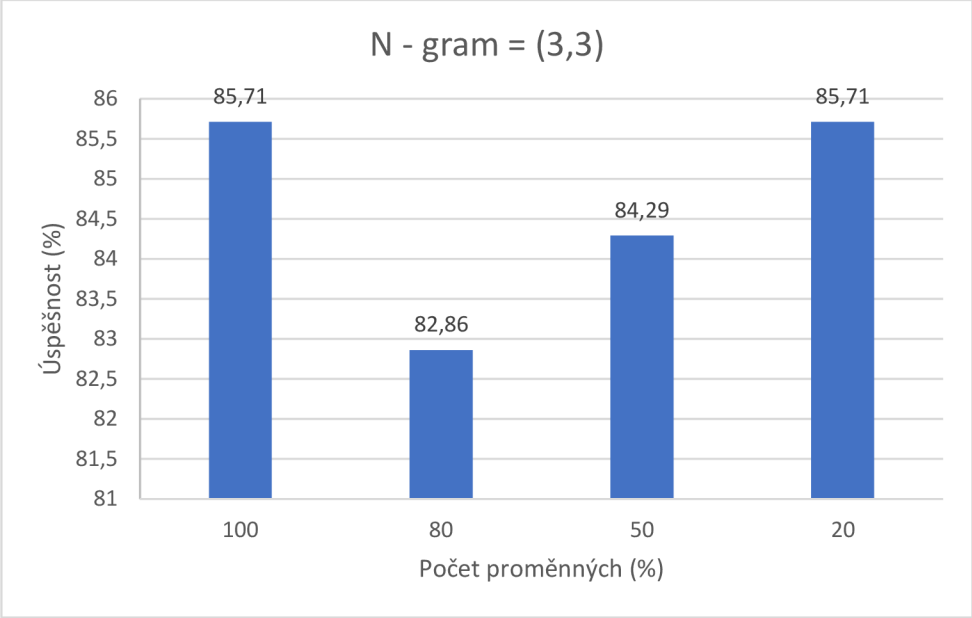
DataFrame	Tabulková forma zápisu dat v Pandasu
Keras	Knihovna poskytující rozhraní pro TensorFlow
KNN	K nejbližších sousedů
N-gram	Počet po sobě jdoucích slov vektorizovaný jako jeden celek
NLTK	Sada knihoven pro zpracování přirozeného jazyka
Pandas	Knihovna pro analýzu dat
QRTs	Quantitative reporting templates
Scikit-learn	Knihovna pro strojové učení v jazyku Python
SCR	Solventnostní kapitálový požadavek
SFCR	Zpráva o solventnosti a finanční situaci pojišťovny
SVM	Podpůrné vektorové stroje
TensorFlow	Knihovna pro umělou inteligenci a neuronové sítě

9 Přílohy

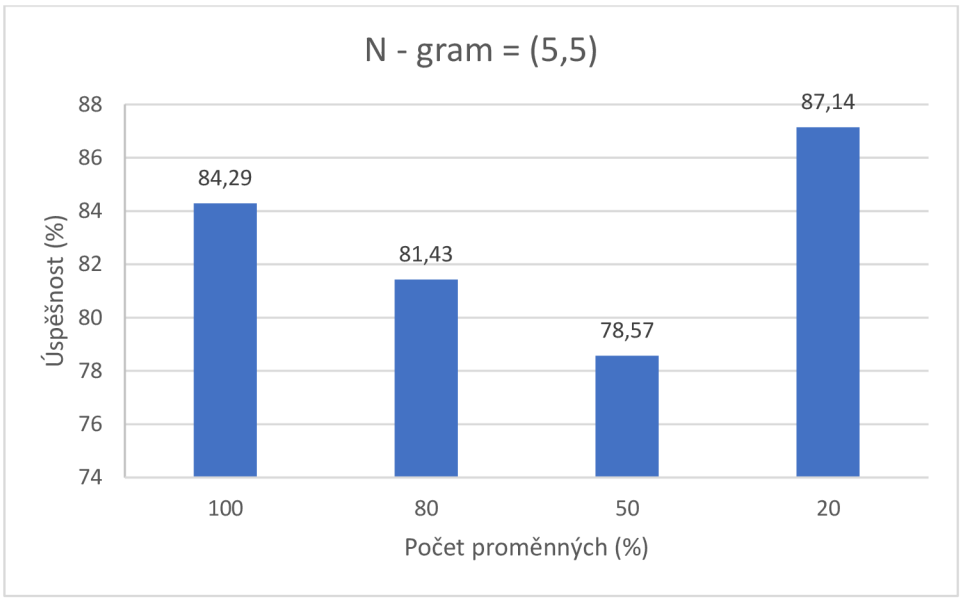
9.1 Grafy výsledků

9.1.1 KNN model pro ostatní N-gramy

Obrázek 33 - KNN pro N-gram = 3

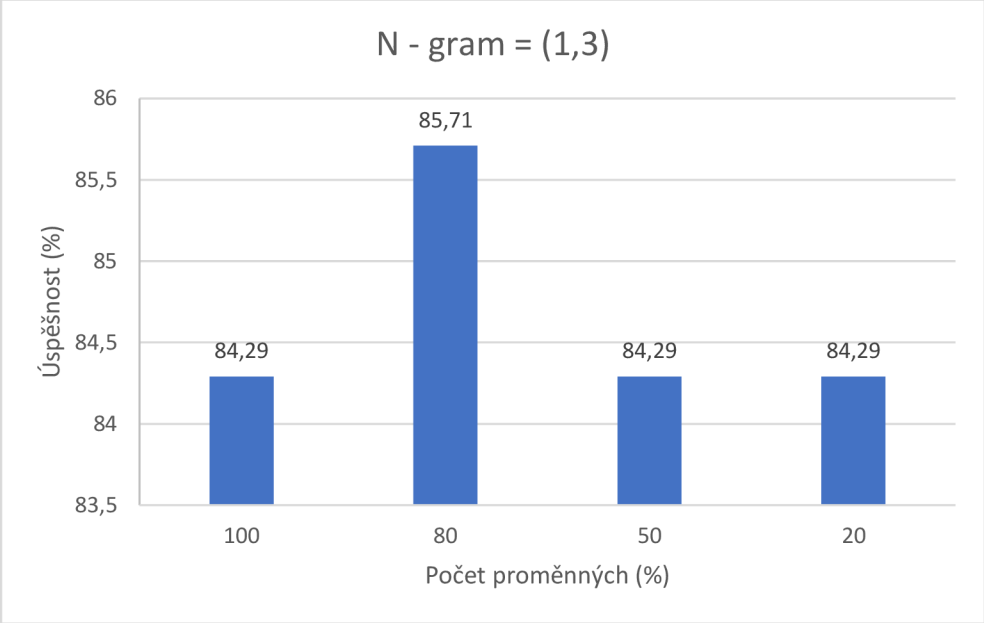


Obrázek 34 - KNN pro N-gram = 5

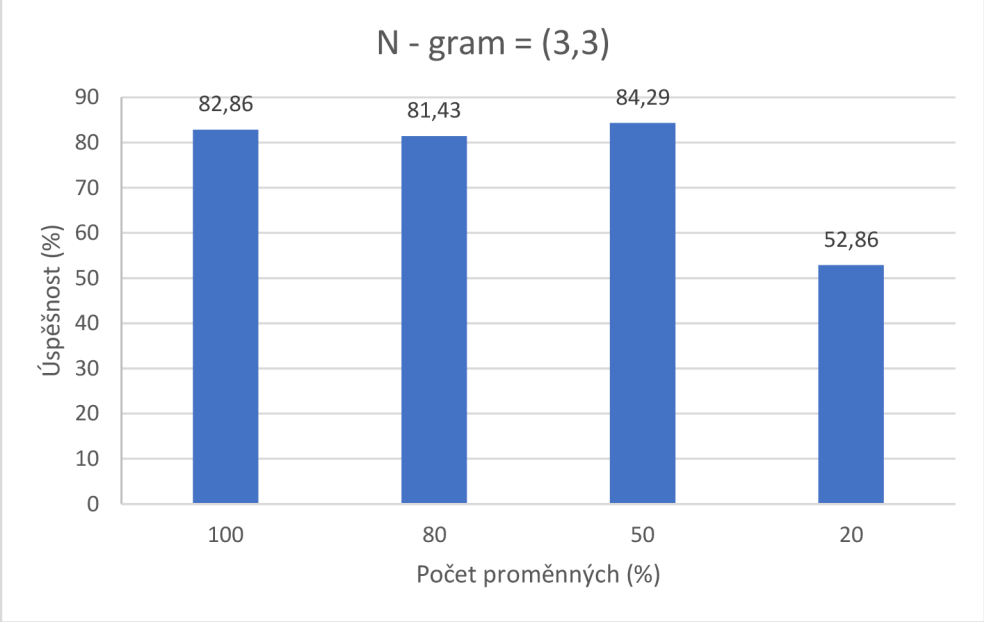


9.1.2 SVM model pro ostatní N-gramy

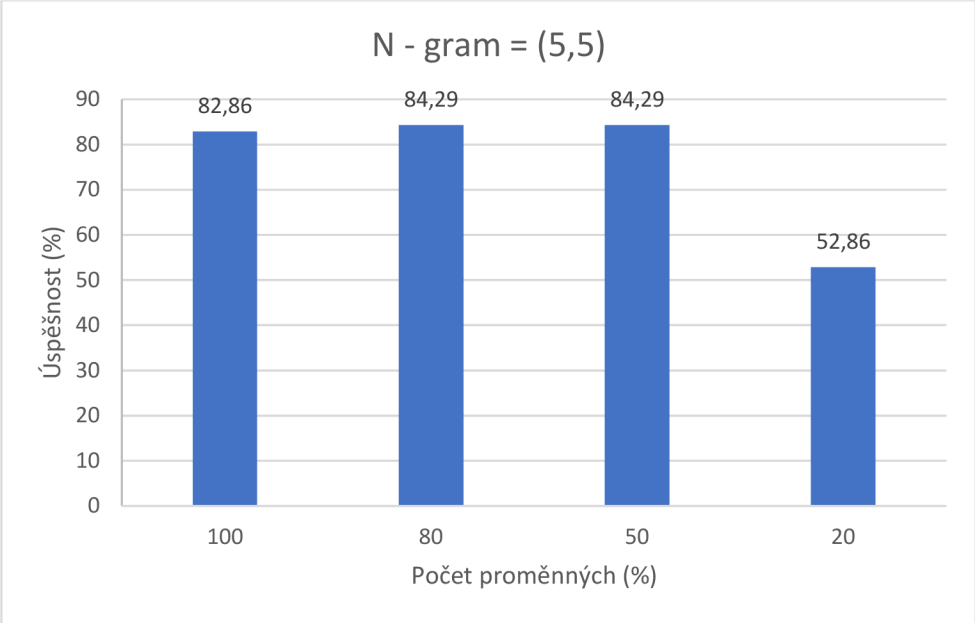
Obrázek 35 - SVM pro N-gram = (1, 3)



Obrázek 36 SVM pro N-gram = 3

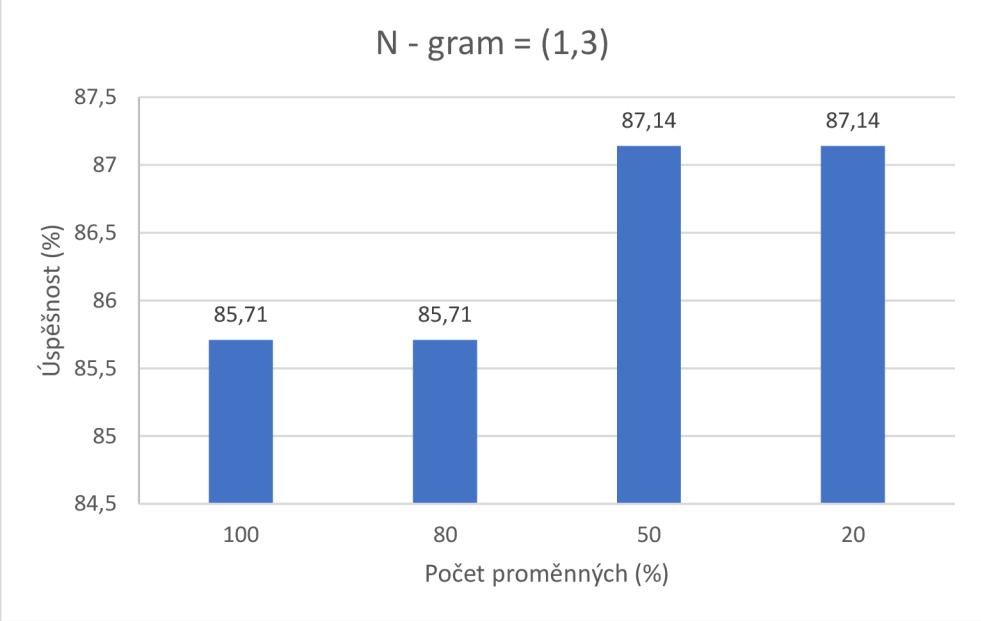


Obrázek 37 - SVM pro N-gram = 5

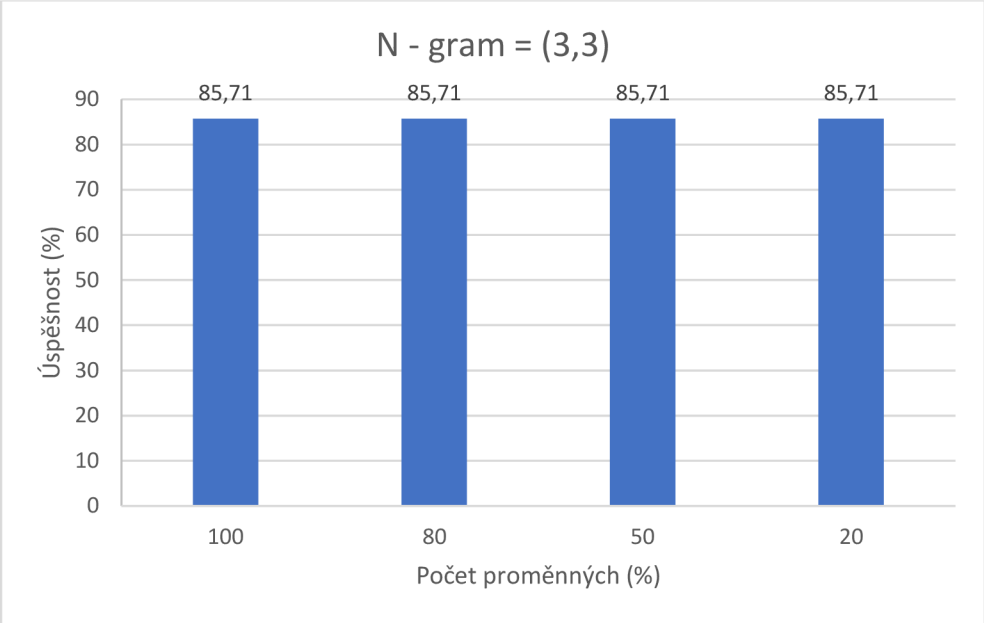


9.1.3 Logistická regrese pro ostatní N-gramy

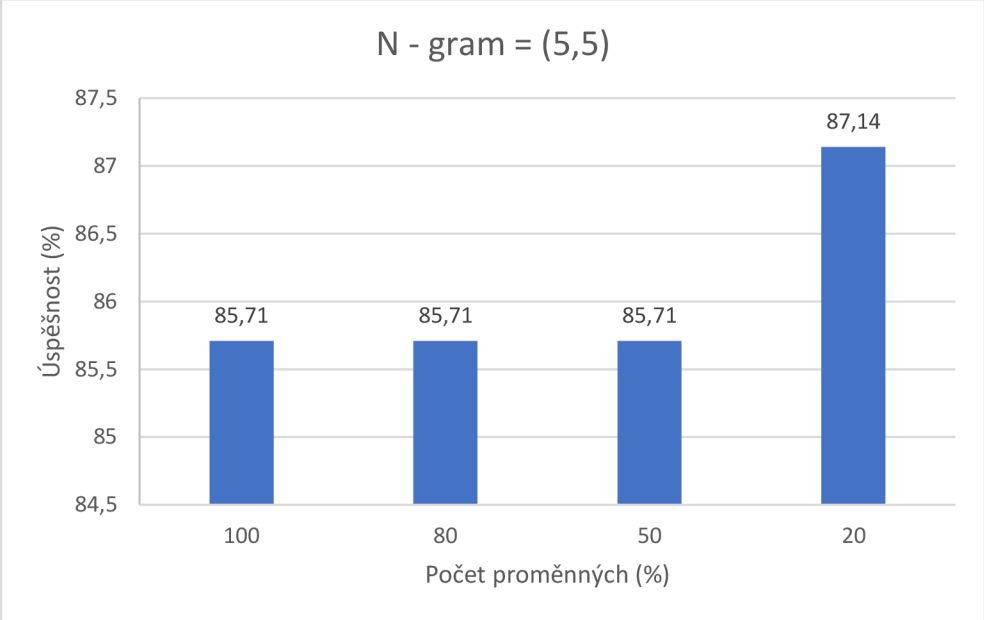
Obrázek 38 - Logistická regrese pro N-gram = (1,3)



Obrázek 39 - Logistická regrese pro N-gram = 3

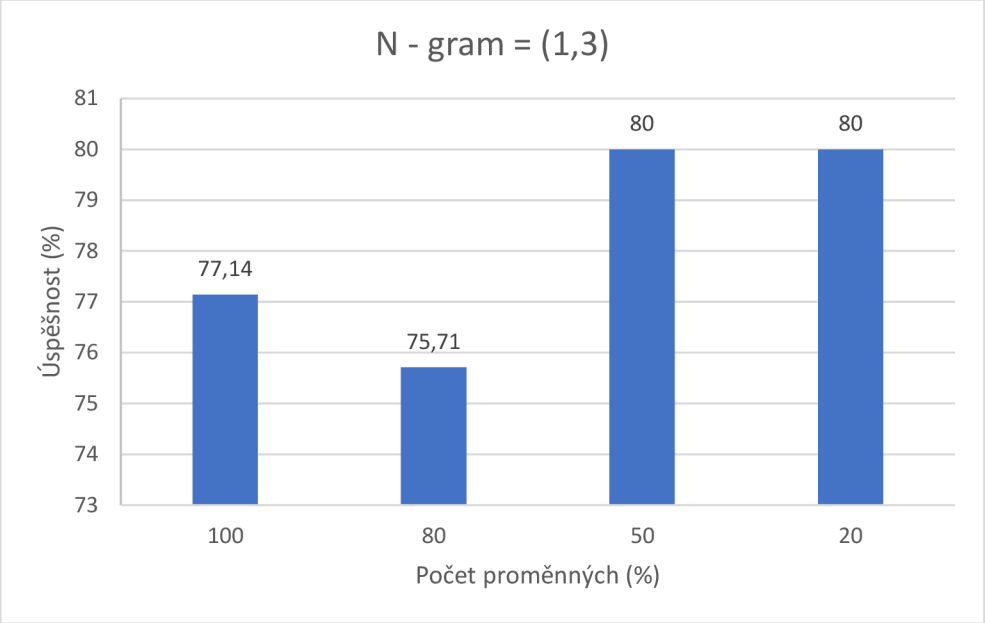


Obrázek 40 - Logistická regrese pro N-gram = 5

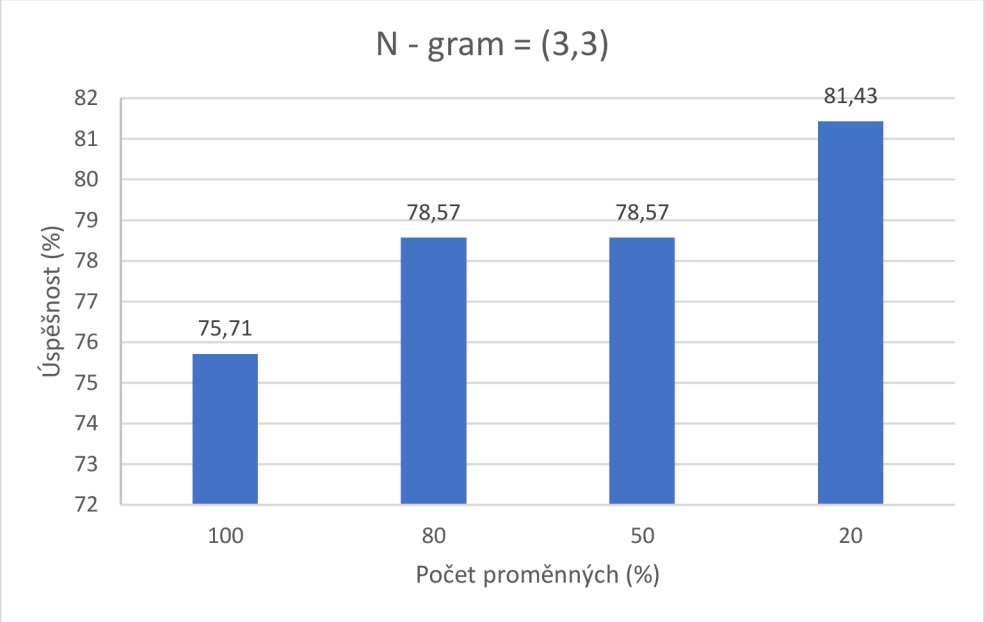


9.1.4 Rozhodovací stromy pro ostatní N-gramy

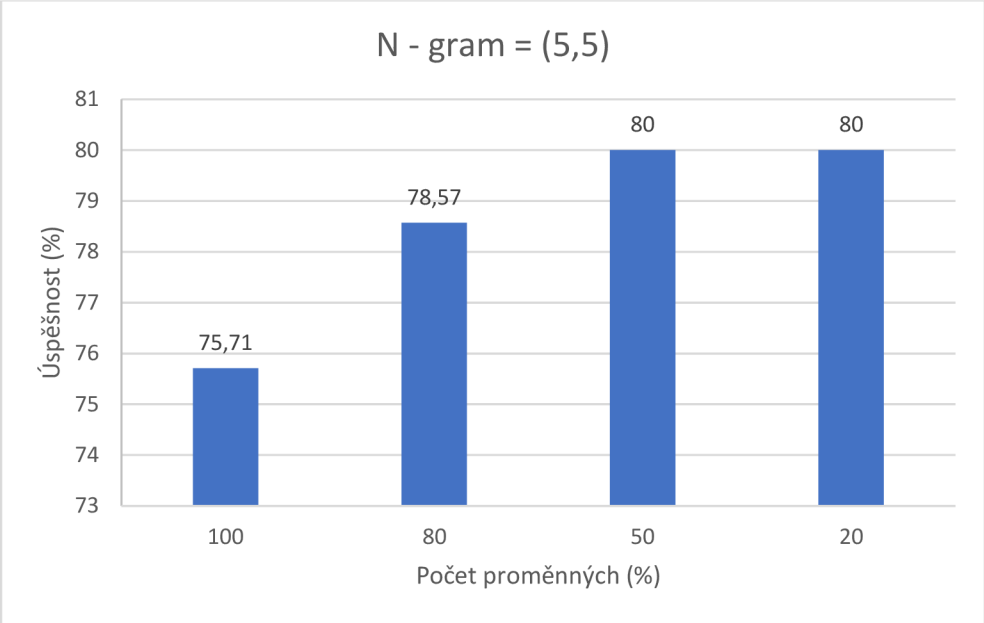
Obrázek 41 – Rozhodovací strom pro N-gram = (1,3)



Obrázek 42 - Rozhodovací strom pro N-gram = 3

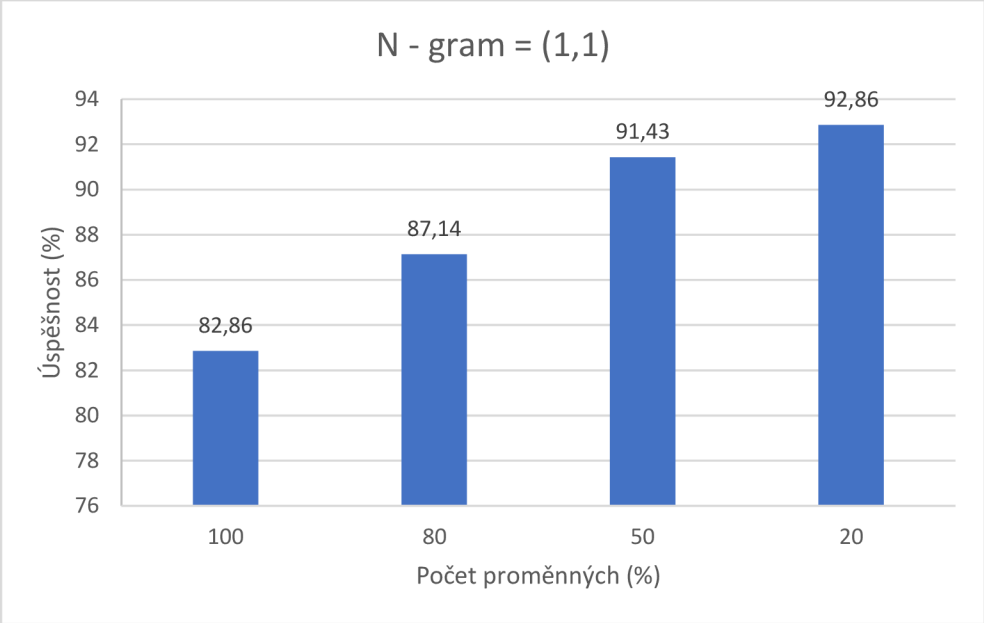


Obrázek 43 - Rozhodovací strom pro N-gram = 5

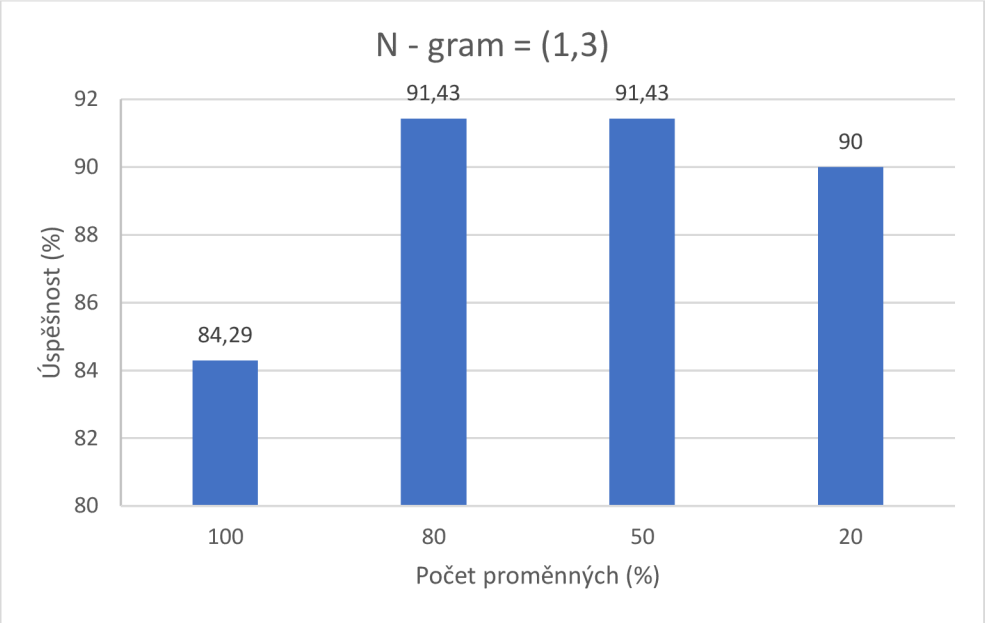


9.1.5 Naivní Bayes pro ostatní N-gramy

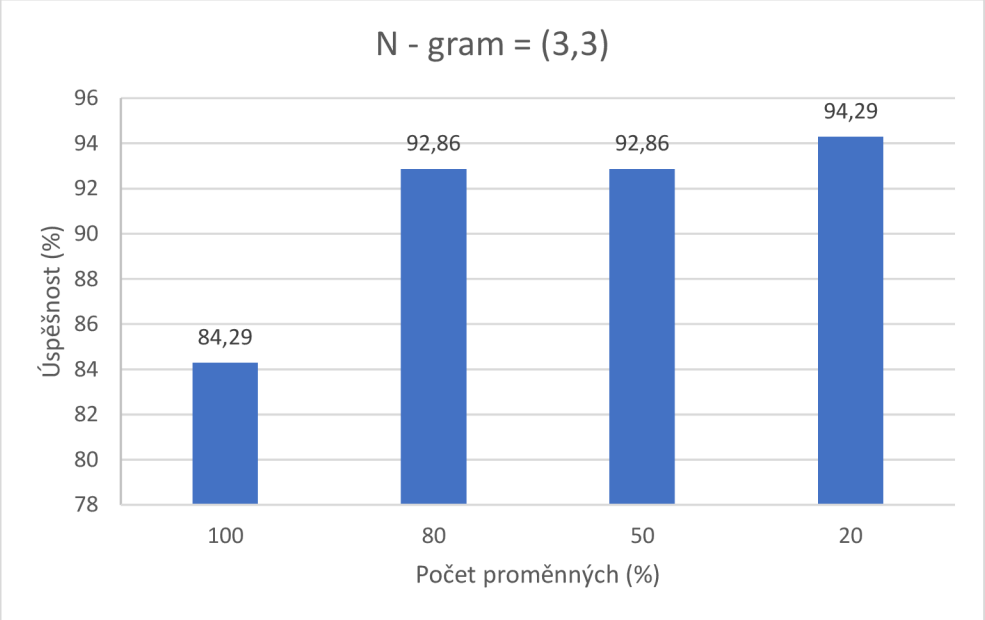
Obrázek 44 – Naivní Bayes pro N-gram = 1



Obrázek 45 - Naivní Bayes pro N-gram = (1,3)

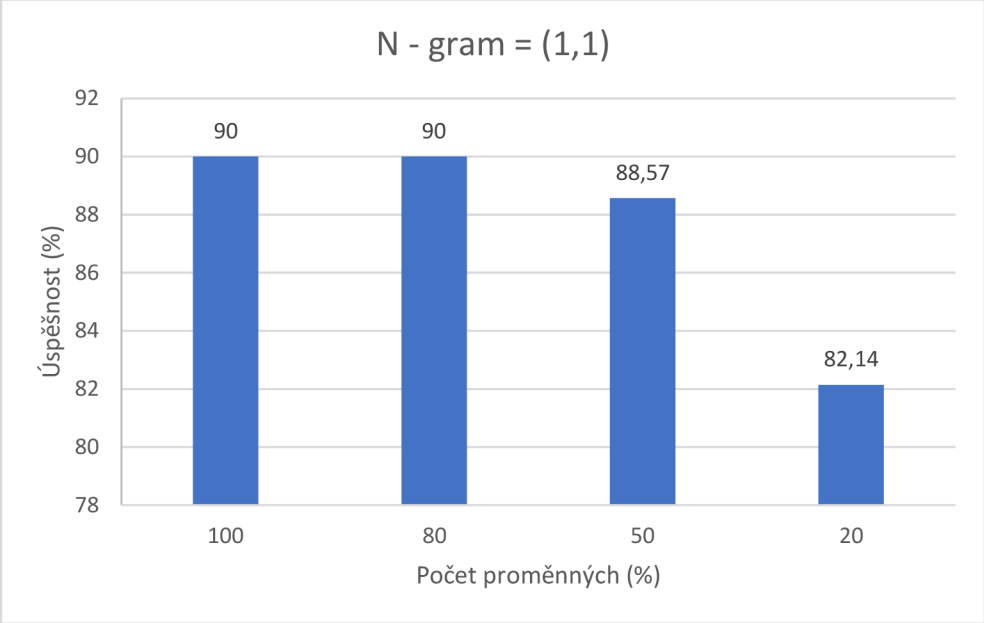


Obrázek 46 - Naivní Bayes pro N-gram = 3

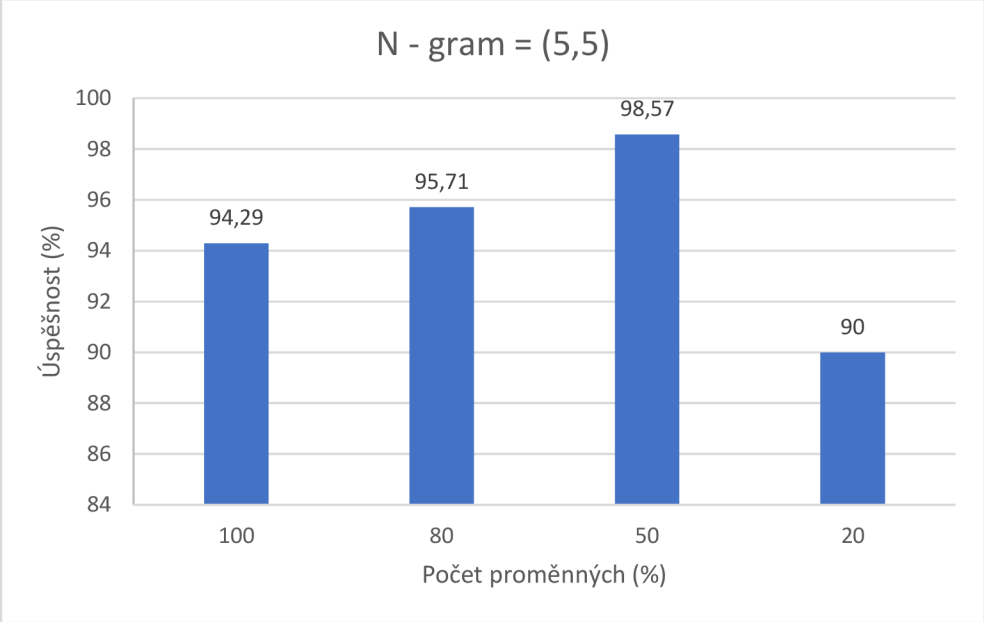


9.1.6 Neuronové sítě pro ostatní N-gramy

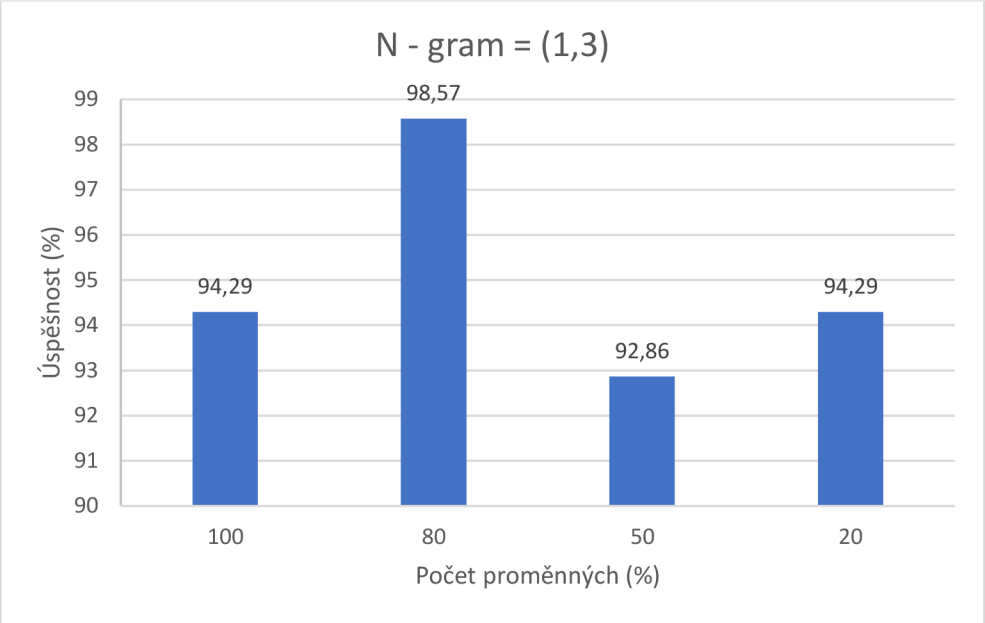
Obrázek 47 – Neuronová síť pro N-gram = 1



Obrázek 48 - Neuronová síť pro N-gram = 5



Obrázek 49 - Neuronová síť pro N-gram = (1,3)



Obrázek 50 - Nastavení modelu pro N-gram(1,3), 4/5 proměnných

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_33 (Dense)             (None, 128)               54226048

dense_34 (Dense)             (None, 64)                8256

dense_35 (Dense)             (None, 1)                 65

=================================================================
Total params: 54,234,369
Trainable params: 54,234,369
Non-trainable params: 0
_________________________________________________________________
```

9.2 Zdrojový kód

Zdrojový kód je psán na platformě JupyterLab. V přiloženém kódu značí mezera mezi řádky novou buňku. V případě dlouhého řádku, zejména komentářů, je řádek rozdělen na dva řádky bez zahrnutí odsazení.

Vstupní dokumenty ve formátu PDF jsou uloženy v adresáři zdrojového kódu ve složce ´reports´ a podsložkách s číslem reportovacího roku. Kvantitativní data jsou uložena ve stejném adresáři v excelovém souboru a rozdělena do listů opět dle čísla reportovacího roku.

9.2.1 Kód importování textu

```
import pandas as pd # to use pd Dataframe
import os # to get ahold of the basic operating system functionalities
from glob import glob # to get the filepaths
from constance import table_of_contents #Table of content is just dictionary with possible headings for Table of contents
for Czech Lang.
import re # to use regular expressions
import pdfplumber #To import text from PDF to pandas Dataframe
import numpy as np #To do basic math operation
import matplotlib.pyplot as plt # to plot the graphs
import pickle # to open and load with pickle
import string #To remove punctation
import collections # To collect single occurrence words
pd.options.display.max_colwidth=100

reporting_years=[2019, 2020, 2021]

sfcrs = dict() # Sfcr dict with key reporting_year stores filepaths
for reporting_year in reporting_years:
    sfcrs[reporting_year] = glob(('\\').join([os.path.normpath('MyPath), 'reports', str(reporting_year), '*.pdf']))

file = 'MyPath' # creating text_splits folder and setting up the path
try:
    os.mkdir(os.path.normpath(os.path.dirname(file) + '\\text_splits'))
except OSError:
    pass


# extracting the text to be stored in dataframes
extracted_text = dict()
number_of_pages=[]
try:
    with open ((os.path.normpath(os.path.dirname(file) + '\\text_splits\\number_of_pages.pkl')), 'rb') as f:
            number_of_pages = pickle.load(f)
except FileNotFoundError:
    pass
```

```
for reporting_year in reporting_years:
    try:
        extracted_text[reporting_year] = pd.read_pickle(os.path.normpath(os.path.dirname(file) +
'\\text_splits\\extracted_text_(' + str(reporting_year) + ').pkl')).fillna(value=np.nan)
    except FileNotFoundError:
        extracted_text[reporting_year] = pd.DataFrame()
        for filepath in sfcrs[reporting_year]:
            toc_page = 0
            undertaking_code = int(re.findall('\d+', filepath)[-1])
            with pdfplumber.open(filepath) as pdf_file:
                try:
                    number_of_pages.append(len(pdf_file.pages))
                    for page in pdf_file.pages:
                        if re.search(table_of_contents['cz'].replace(' ', '.*'), page.extract_text(), re.IGNORECASE):
                            toc_page = page.page_number - 1 #If the text is found in table_of_contents['cz'], the page number is
stored in the toc_page variable and the loop is broken.
                            break
                except TypeError:
                    pass
                actual_page = toc_page
                if toc_page:
                    toc_numbers = re.findall('\d+', pdf_file.pages[toc_page].extract_text(), flags=re.IGNORECASE) #This part
of the code is checking the page numbers listed in the table of contents, and using the last page number listed to
determine the starting page for processing the text in the PDF file.
                    if toc_numbers and (int(toc_numbers[-1])-1 <= toc_page):
                        actual_page = int(toc_numbers[-1])-1
                try:
                    for page in pdf_file.pages[actual_page:]:
                        extracted_text[reporting_year].at[actual_page, undertaking_code] = page.extract_text()
                        actual_page += 1
                except TypeError:
                    pass
        for undertaking_code in extracted_text[reporting_year]:
            if ((extracted_text[reporting_year][undertaking_code].values == '').sum() +
extracted_text[reporting_year][undertaking_code].isna().sum()) == len(extracted_text[reporting_year].index):
                extracted_text[reporting_year].drop(undertaking_code, axis=1, inplace=True)
        extracted_text[reporting_year].to_pickle(os.path.normpath(os.path.dirname(file) + '\\text_splits\\extracted_text_(' +
str(reporting_year) + ').pkl'))
with open((os.path.normpath(os.path.dirname(file) + '\\text_splits\\number_of_pages.pkl')),'wb')  as f:
    pickle.dump(number_of_pages, f)    #If we run this cell pickle will overwrite this again by same pickle
print(number_of_pages)
```

9.2.2 Kód vytváření Dataframu a průzkumové analýzy

```
#Graph of pages numbers
pages_maximum=np.max(number_of_pages)
pages_minimum=np.min(number_of_pages)
average_pages=np.mean(number_of_pages)
print(pages_maximum, pages_minimum, average_pages)
plt.figure(figsize=(8, 5))
bars=plt.bar(['maximum', 'průměr', 'minimum'], [pages_maximum, average_pages, pages_minimum], width=0.4)
plt.ylabel('Počet stran')
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height/2, '%d'  %int(height),
            ha='center', va='center')
plt.show()



#Making dictionary with quant features and target feature
quantitative_feature_str = { }
for reporting_year in reporting_years:
    file_path = "Quantitative_features.xlsx" # Construct the file path
    quantitative_feature_str[reporting_year] = pd.read_excel(file, sheet_name=str(reporting_year)) # Load the sheet with
the given reporting year into a dataframe
quantitative_feature_str[reporting_year]=quantitative_feature_str[reporting_year].drop(columns='Solvency_ratio_from_n
ext_year')
    quantitative_feature_str[reporting_year].set_index('Pojistovna_ID', inplace=True)



#Deleting the features thata have more then 40% of NaN values
import warnings
warnings.filterwarnings('ignore')
quantitative_feature_all_years = pd.DataFrame(columns=quantitative_feature_str[reporting_years[0]].columns)
for reporting_year in reporting_years:
    df_temp = quantitative_feature_str[reporting_year] #temporary dataframe for each reporting year
    quantitative_feature_all_years = quantitative_feature_all_years.append(df_temp) #merge all the years together
percentage_nan_values_all_years = quantitative_feature_all_years.isna().mean(axis=0) * 100
columns_to_drop = percentage_nan_values_all_years[percentage_nan_values_all_years > 40].index.tolist()
for reporting_year in reporting_years:
    quantitative_feature_str[reporting_year] = quantitative_feature_str[reporting_year].drop(columns=columns_to_drop)
#dropping columns
```

```
#Getting the list of quantitative columns to be use later in mixed model
columns_of_quant_features = list(quantitative_feature_str[next(iter(quantitative_feature_str))].columns)
print(columns_of_quant_features)



#Creating full_text dataframe that merge all the reporting years together and also joining text and quantitative features
full_only_text = pd.DataFrame(columns=['Undertaking Code', 'Text'])

for year, values in extracted_text.items():
    if year in quantitative_feature_str.keys():  # check if there are quantitative features for this year
        df_qf = quantitative_feature_str[year]  # get the quantitative feature dataframe for this year
        for undertaking_code, texts in values.items():
            texts = [str(text) for text in texts]
            if undertaking_code in df_qf.index:  # check if there are quantitative features for this undertaking code
                row = {'Undertaking Code': str(undertaking_code) + '_' + str(year), 'Text': " ".join(texts)}
                row.update(df_qf.loc[undertaking_code].to_dict())  # add quantitative features to the row
                full_only_text = full_only_text.append(row, ignore_index=True)

full_text = full_only_text.set_index('Undertaking Code')



full_text[columns_of_quant_features] =
full_text[columns_of_quant_features].fillna(full_text[columns_of_quant_features].mean()) #Filling missing values in
indicator columns by mean
```

9.2.3 Kód předzpracování textu a vektorizace

```
# Opening text file containing stopwords
with open('stopwords.txt', 'r', encoding='UTF-8') as file_stopwords:
    text = file_stopwords.read()
stop_words = text.split()



import nltk
nltk.download('punkt')

#Remove Punctation
def remove_punctuation(txt):
    text_nopunct ="".join([c for c in txt if c not in string.punctuation])
    return text_nopunct

#Tokenize text using nltk library
```

```
def tokenize(txt):
    tokens = nltk.word_tokenize(txt)
    return tokens

#Remove tokens that include numbers
def to_lower(tokens):
    lower_letters = [token.lower() for token in tokens]
    return lower_letters

#Remove words with less than 3 letters and including numbers
def remove_short(tokens):
    text_no_stop = [token for token in tokens if (len(token) > 2) and (not any(char.isdigit() for char in token))]
    return text_no_stop

#Remove stopwords stored in txt file
def remove_stopwords(tokens, stop_words):
    text_no_stop=[token for token in tokens if token not in stop_words]
    return text_no_stop

#Cut all words for first five letters
def first_five_letters(token_list):
    first_five = [token[:5] for token in token_list]
    return first_five

full_text['Text_nopunct']=full_text['Text'].apply(lambda x: remove_punctuation(x))
full_text['Text_tokenized']=full_text['Text_nopunct'].apply(lambda x: tokenize(x))
full_text['lower_only']=full_text['Text_tokenized'].apply(lambda x: to_lower(x))
full_text['No_single_letter']=full_text['lower_only'].apply(lambda x: remove_short(x))
full_text['No_stop_words']=full_text['No_single_letter'].apply(lambda x: remove_stopwords(x, stop_words))
full_text["five_letters"] = full_text["No_stop_words"].apply(lambda x: first_five_letters(x))
text_list = full_text["five_letters"].apply(lambda x: " ".join(x)).tolist()

text_list_no_stop = full_text['No_stop_words'].apply(lambda x: " ".join(x)).tolist()
text_no_stop = ' '.join(text_list_no_stop)
words = text_no_stop.split() # Split the string into words
unique_words = set(words) # Get unique words
print("number of uniqe words is: ",(len(unique_words)))
print("number of words after deleting stop words is: ",len(words))

text_list_lower_only = full_text['lower_only'].apply(lambda x: " ".join(x)).tolist()
text_no_stop = ' '.join(text_list_lower_only)
words_before_cleaning = text_no_stop.split() # Split the string into words
unique_words_before_cleaning = set(words_before_cleaning) # Get unique words
```

```
print("number of uniqe words before cleaning is: ",(len(unique_words_before_cleaning)))
print("number of  total words before cleaning is: ",len(words_before_cleaning))

text_no_stop_five_lett = ' '.join(text_list)
words = text_no_stop_five_lett.split()
unique_words = set(words)

words = text_no_stop_five_lett.split() # Split the string into words
word_counts = collections.Counter(words) # Count the occurrences of each word
one_word_occurrence = [word for word, count in word_counts.items() if count == 1]

#Loading or crating text list ready excluded one word occurence words
try:
    with open ((os.path.normpath(os.path.dirname(file) + '\\text_splits\\text_list_ready.pkl')), 'rb') as f:
        text_list_ready = pickle.load(f)
except:
    text_list_ready=[]
    for text in text_list:
        words = text.split()
        filtered_words = [word for word in words if word not in one_word_occurrence]
        filtered_text = ' '.join(filtered_words)
        text_list_ready.append(filtered_text)
    with open((os.path.normpath(os.path.dirname(file) + '\\text_splits\\text_list_ready.pkl')),'wb')  as f:
        pickle.dump(text_list_ready, f)



from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(ngram_range=(1,1))  #Here enter the desired N-grams
tfidf.fit(text_list_ready) # Fit the vectorizer to the text data
tfidf_matrix = tfidf.transform(text_list_ready) # Transform the text data into a TF-IDF matrix

tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf.get_feature_names(), index=full_text.index) # Convert the
matrix to a DataFrame

tfidf_df_final = tfidf_df.join(full_text[columns_of_quant_features])
X = tfidf_df_final[tfidf_df_final.columns.difference(['Solvency_ratio_bucket'])]
y = tfidf_df_final['Solvency_ratio_bucket']

#In case of splitting dataframe into training and testing dataset use following code snippet:
tfidf_df_final_test=tfidf_df_final.iloc[48:,]
tfidf_df_final=tfidf_df_final.iloc[:48,] #Use first 48 rows for training
X = tfidf_df_final[tfidf_df_final.columns.difference(['Solvency_ratio_bucket'])]
y = tfidf_df_final['Solvency_ratio_bucket']
```

```
X_test = tfidf_df_final_test[tfidf_df_final_test.columns.difference(['Solvency_ratio_bucket'])]
y_test = tfidf_df_final_test['Solvency_ratio_bucket']

#Feature selection using SelectPercentile for only training set:
subset_text_list_ready = text_list_ready[:48]
subset_tfidf_matrix = tfidf.transform(subset_text_list_ready)
subset_tfidf_df = pd.DataFrame(subset_tfidf_matrix.toarray(), columns=tfidf.get_feature_names())
```

9.2.4 Kód pro jednotlivé modely a selekce proměnných

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, make_scorer
from sklearn.model_selection import cross_val_score

X = tfidf_df_final[tfidf_df_final.columns.difference(['Solvency_ratio_bucket'])]
y = tfidf_df_final['Solvency_ratio_bucket']
knn= KNeighborsClassifier() #Here pass the target model for SVM: svm_model = SVC(kernel='linear') , for naive bayes:
nb=GaussianNB(), …
accuracy = make_scorer(accuracy_score)
accuracy_scores = cross_val_score(knn, X, y, scoring = accuracy, cv = 7)
print("accuracy score:", accuracy_scores)
print("accuracy score:", accuracy_scores.mean())
print("Accuracy std:", np.std(accuracy_scores))

#Feature importance using SelectPercentile
from sklearn.feature_selection import SelectPercentile, f_classif, SelectFromModel
percentile = 20 # Select the top X % of features based on ANOVA F-value
selector = SelectPercentile(f_classif, percentile=percentile)
selector.fit(tfidf_matrix, y)
tfidf_selected = selector.transform(tfidf_matrix)
selected_indices = selector.get_support(indices=True) # Get the indices of the selected features
tfidf_df_selected = tfidf_df.iloc[:, selected_indices]
tfidf_df_final = tfidf_df_selected.join(full_text[columns_of_quant_features])
```

9.2.5 Kód pro model neuronové sítě

```
y=y.replace({'positive': 1, 'stable': 0})
X = X.to_numpy()
```

```
y = y.to_numpy()

from sklearn.model_selection import KFold
from tensorflow import keras


model = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(X.shape[1],)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
k = 7
kf = KFold(n_splits=k)
accuracy_scores = []
for train_idx, val_idx in kf.split(X):
    X_train, y_train = X[train_idx], y[train_idx]
    X_val, y_val = X[val_idx], y[val_idx]
    history = model.fit(X_train, y_train, epochs=10, batch_size=128, validation_data=(X_val, y_val))
    accuracy_scores.append(history.history['val_accuracy'][-1])
print("Accuracy score:", accuracy_scores)
print("Accuracy score:", np.mean(accuracy_scores))
print("Accuracy std:", np.std(accuracy_scores))
```

9.2.6 Kód pro grid search

```
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.model_selection import LeaveOneOut

knn= KNeighborsClassifier() #Here pass the target model
param_grid = {'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 13, 21, 34], 'weights': ['uniform', 'distance'], 'p': [1, 2, 3]} # Here pass
the dictionary with parameters for target model

grid_search = GridSearchCV(knn, param_grid, cv=LeaveOneOut())
grid_search.fit(X, y)

print("Best hyperparameters:", grid_search.best_params_)
accuracy_scores = cross_val_score(grid_search.best_estimator_, X, y, cv=LeaveOneOut())

print("Accuracy score:", accuracy_scores)
print("Mean accuracy:", np.mean(accuracy_scores))
print("Accuracy std:", np.std(accuracy_scores))
```