Mendelova univerzita v Brně
Provozně ekonomická fakulta

Automatická kategorizace zákaznických požadavků

Diplomová práce

Vedoucí práce:
doc. Ing. František Dařena, Ph.D.

Bc. Filip Koukal

Brno 2022

NA TOMTO LISTU SE NACHÁZÍ ORIGINÁLNÍ ZADÁNÍ PRÁCE

Poděkování

Děkuji doc. Ing. Františku Dařenovi, Ph.D za cenné rady, připomínky a trpělivost při vedení práce.

Čestné prohlášení

Prohlašuji, že jsem práci **Automatická kategorizace zákaznických požadavků** vypracoval samostatně a veškeré použité prameny a informace uvádím v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a v souladu s platnou Směrnicí o zveřejňování závěrečných prací.

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 6. května 2022 .....

podpis

Abstract

Koukal, F. Automatic categorization of customer requests. Brno, 2022.

This thesis focuses on the creation of a system for automatic categorization of customer requests written in natural languages. Three main experiments have been defined: category assignment for a newly created request, solver recommendation for a newly created request, and finding and marking duplicate requests. A variety of approaches were tested in each experiment, including traditional natural language processing techniques and new breakthroughs. A service implementing the best approaches was subsequently created and deployed to the Microsoft Azure cloud platform.

Key words: text mining, machine learning, classification, transformer, BERT, customer support, helpdesk

Abstrakt

Koukal, F. Automatická kategorizace zákaznických požadavků. Brno, 2022.

Tato diplomová práce se zabývá vybudováním systému pro automatickou kategorizaci zákaznických požadavků psaných přirozeným jazykem. V rámci práce jsou definovány tři hlavní experimenty zkoumající různé přístupy pro přiřazení kategorie k požadavku, doporučení řešitele k požadavku a vyhledání a označení duplicitních požadavků. V rámci experimentů jsou srovnány tradiční přístupy zpracování přirozeného jazyka a moderní průlomy. Z nejlepších přístupů je poté vytvořena služba a nasazena na cloudové platformě Microsoft Azure.

Klíčová slova: dolování z textu, strojové učení, klasifikace, transformer, BERT, zákaznická podpora, helpdesk

Obsah

1 Úvod a cíl práce

1.1 Úvod do problematiky

Zákaznické požadavky a stížnosti jsou věci, s kterými se setká téměř každá firma. Ať už se jedná o dotazy o produktech, stížnosti či reklamace, firma, která si chce se svými zákazníky budovat kvalitní vztah, na ně musí efektivně a v přiměřeném čase reagovat. Ukazatel, který tuto reakci dokáže značně zefektivnit, je kategorie daného požadavku – o čem zákazník mluví.

Nicméně už zde narážíme na problémy, jak zjistit, co od nás zákazník požaduje? Tento problém se dá řešit dvěma hlavními způsoby, buď se zákazníka explicitně dotážeme při vzniku požadavku, anebo se pokusíme jeho úmysl zrekonstruovat ze vstupů, které nám poskytl. Oba postupy mají své výhody a nevýhody, například jak si můžeme být jistí, že zákazník správně označil kategorii svého požadavku. Co když svou kategorii přehlídl, nebo omylem zvolil jinou? Zkrátka se zde objevují nežádoucí situace, které nemůžeme ovlivnit. U druhého postupu opět narážíme na problémy, především na otázky: Jak úmysl zákazníka zrekonstruovat? Z jakých vstupů? Kdo to bude dělat? Naštěstní na všechny tyto otázky má moderní výpočetní technika odpovědi ve formě automatizace, strojového učení a umělé inteligence.

COVID-19 pandemie ukázala, že budování robustních online prostředků je klíčové pro zefektivnění vzdálené komunikace. Podle průzkumu KPMG (2020) bylo zjištěno, že více než 80 % amerických firem se připravuje investovat do moderních výpočetních technologií stavěných na umělé inteligenci a strojovém učení. Z toho více než polovina plánuje migraci svých výpočetních procesů do cloudových platforem. Je tedy zřejmé, že digitalizace a automatizace klíčových podnikových procesů s využitím strojového učení je v dnešní době jedna z nejžádanějších oblastí výpočetní techniky. Tento jev je dále podporován faktem, že tato práce je úzce spojena s projektem modernizace helpdesku informačního systému firmy ALVAO s.r.o., kde je mezi hlavními požadavky stanoveno využití cloudové platformy Microsoft Azure pro účely kategorizace uživatelských požadavků.

Již zmíněné vstupy neboli data jsou často v textové formě. Jedná se například o zprávu z live chatu, titulek nebo tělo emailu. Vesměs jsou to prostředky přirozeného jazyka, který nemá jasnou strukturu, z které by se dal snadno určit úmysl zákazníka. Je tedy nutné data nějakým způsobem zpracovat do podoby, u které lze jednoznačně stanovit úmysl. Přesně těmito daty a jejich zpracováním se diplomová práce zabývá.

Oblast analýzy přirozeného jazyka má už dlouhou historii a neustále se objevují nové příležitosti, kde je možné tuto disciplínu uplatnit. Moderní svět internetu a online propojení neustále přináší vhodné aplikace, které často značně transformují uživatelské zkušenosti s různými produkty pro kvalitnější a efektivnější zážitek. Při dlouhodobé relevanci lze očekávat pravidelné průlomy a inovaci v jakékoliv oblasti a tato se nijak neliší.

V posledních letech zažila oblast zpracování přirozeného jazyka několik průlomů, které umožnují řešit tradiční problémy s novou úrovní přesnosti a škálovatelnosti. Tento fakt pro nás teď, více než jindy, otevírá příležitosti pro zefektivnění a zkvalitnění komunikace se zákazníkem a pro zvýšení pracovního výkonu firemních operátorů.

1.2 Cíl práce

Cílem práce je vybudovat systém, který bude automatizovaně kategorizovat zákaznické požadavky a dotazy psané přirozeným jazykem. Jako vstup pro tvorbu klasifikátoru budou sloužit existující dotazy. Součástí práce bude vytvoření vhodné učicí množiny dat, zvolení vhodné reprezentace různých typů vstupů, definování vhodných experimentů pro predikci významných vlastností požadavků, definování vhodných způsobů rozhodování o zařazení požadavku do jedné nebo více kategorií a následné vytvoření aplikace implementující definované způsoby.

2 Současný stav problematiky

2.1 Zákaznické požadavky

Zákaznická podpora je aktivita, kterou firma poskytuje před, v průběhu a po prodeji služby či produktu. Jedná se o různé formy podpory zajišťující spokojenost zákazníka v jakékoliv fázi prodeje (Menken a Blokdijk, 2009).

V mnohých případech je struktura zákaznické podpory organizována ve formě tzv. helpdesku, který poskytuje jednotný, dobře viditelný bod kontaktu se zákazníky. Požadavky bývají v helpdesk systémech reprezentovány tzv. tikety, které zachycují veškerou interakci uživatele s operátorem ve formě konverzace. V rámci konverzace uživatel postupně upřesňuje svůj problém, kde operátor reaguje na zprávy uživatele dodatečným dotazováním, poskytováním řešení či přesměrování požadavku do vhodného oddělení. Tikety obvykle mají přiřazené různé stavy signalizující jejich životní cyklus. Stavy mohou zahrnovat například nový tiket čekající na operátora, aktuálně řešený tiket, přiřazení řešení, čekání na řešitele, vyřešený tiket apod. Informace o změně stavu tiketu a dalších významných manipulací bývají často poskytnuty uživateli prostřednictvím systémových zpráv, které jsou poslány v rámci tiketu automatizovaně (Martin, 2019).

Jakmile se tikety dostanou do určitého stavu, kde operátor má dostatečné množství informací, tak je k nim přiřazen řešitel, který má za úkol požadavek efektivně vyřešit. Požadavky obsahující podobné či stejné problémy se často slučují, jelikož je důležité, aby nenastala situace, kde dva řešitelé řeší stejný problém, což produkuje různá nedorozumění, problémy při aplikaci řešení a neefektivní využití firemních prostředků. Duplicitní tikety je proto vhodné včas odhalit, sloučit a vyřešit souběžně.

Tikety dále bývají uspořádány do front, které přiřazují určitou prioritu každému požadavku pro zamezení stagnace otevřených tiketů. Helpdesky často využívají vícestupňovou strukturu, ve které je první kontakt uživatele s operátorem nejvýznamnější pro včasné určení problému a následné přesměrování tiketu do správného oddělení. Nicméně, aby byl operátor schopný korektně přesměrovat tiket, musí vědět, čeho se požadavek týká. Pro toto rozhodnutí pomáhá kategorie požadavku, kterou uživatel buď zvolí sám při tvorbě požadavku, pokud je mu to umožněno, nebo je operátor nucen přiřadit vhodnou kategorii. V mnohých případech existuje na nejnižší úrovni helpdesku samoobsluha, která obsahuje informace z báze znalostí, pomocí kterých může uživatel vyřešit svůj problém bez zásahu operátora. V případě, kdy je báze znalostí nedostačující pro vyřešení problému, nebo požadavek nemá určenou kategorii, je tiket eskalován do jedné z vyšších úrovní zastoupených operátorem (Martin, 2019).

Helpdesky jako takové nemusí sloužit pouze pro podporu zákazníků, a mnohé firmy je využívají interně pro podporu svých zaměstnanců. Pro interní účely se poté často využívají pro technickou podporu firemních výpočetních prostředků. Vstupní požadavky mohou mít různou formu, například telefonát nebo emailová zpráva,

nicméně výsledný tiket v systému helpdesku se mnohdy projeví v textové podobě (Beisse, 2014).

Kromě samotného textu může tiket obsahovat další informace jako například označení uživatele, který tiket vytvořil, čas vytvoření apod. Nicméně tyto údaje málokdy pomáhají při stanovení kategorie tiketu, a proto je nutné se zaměřit na textovou část požadavku. Neexistují obecné požadavky na délku textové části tiketu, nicméně obvykle se jedná o texty obsahující několik vět popisujících problém uživatele.

Jak už bylo zmíněno, kategorie tiketů bývá dobrý ukazatel pro rychlé zjištění zákazníkova problému. Problematika kategorizace požadavků si tedy zaslouží bližší analýzu z hlediska samotných kategorií. Landsman (2015) zdůrazňuje, že pro efektivitu kategorie tiketů systém nesmí dosahovat počtů odlišných kategorií v řádu stovek. Následně doporučuje vyšší hranici zhruba 20 kategorií s tím, že při vyšším počtu je vhodné využívat podkategorie. Při analýze prací řešící automatickou kategorizaci tiketů od Paramesh et al. (2018), Parmar et al. (2018) a Eichhorn (2020), lze zjistit, že počet kategorií se opravdu pohybuje pod 20, navíc všechny tři práce nevyužívaly podkategorie. Můžeme tedy konstatovat, že se většinou jedná o single-label multiclass klasifikační problém, kde se snažíme přiřadit kategorii k tiketu pro správné přesměrování do jednoho oddělení nebo přiřazení jednomu řešiteli. Tento problém představuje jedno z nejčastějších využití strojového učení. V případě využívání podkategorií tiketů by se jednalo o hierarchickou klasifikaci, která by mohla poskytovat přesnější přesměrování tiketu přímo ke specialistovi.

Nicméně pouze samotné stanovení kategorie tiketu nestačí pro vybudování kvalitního automatizovaného systému. Podle Olson (2018) jsou v oblasti zákaznické podpory doba odezvy operátora a doba vyřešení požadavku jedny z nejdůležitějších klíčových ukazatelů výkonnosti, a proto je nutné přiřadit správnou kategorii tiketu co nejdříve po vytvoření pro minimální zdržení operátorů.

Obecně se dá očekávat, že některé problémy budou mnohem častější než jiné, a z toho lze usoudit, že rozložení kategorií tiketů bude poměrně nevyvážené. Již zmíněné tři práce opět potvrzují tento jev a dvě z nich se snaží kategorie vyrovnat různými způsoby. Tento fakt představuje značné problémy pro algoritmy strojového učení, jelikož je pro natrénování kvalitního klasifikátoru potřeba dostatečné množství dat ze všech kategorií.

2.2 Zpracování přirozeného jazyka

Oblast zpracování přirozeného jazyka, anglicky natural language processing, často zkrácené na NLP, se zabývá porozuměním lidského jazyka na úrovni počítače. Jedná se o komponent umělé inteligence, který se snaží z textových či mluvených dat zjistit klíčové informace a podle nich se dostat k danému závěru. Textová data jsou na rozdíl od tradičních číselných či nominálních hodnot nestrukturovaná, a proto je nutné vstupy značně přetvarovat pro jakoukoliv aplikaci metod výpočetní techniky (Goldberg, 2017).

Tento proces se nazývá předzpracování a tvoří první fázi NLP. Druhá fáze se zabývá postavením algoritmu, který vstupy z první fáze přijme a transformuje je do užitečného výstupu. Tyto výstupy můžou mít několik podob, například téma textu, jeho sentiment nebo jeho klíčové rysy (Feldman a Sanger, 2007).

Metody předzpracování mají na další fáze zpracování textu významný efekt, a proto je nutné každou metodu zvážit s důrazem na aktuálně řešený problém, tedy charakter vstupního textu, používaný algoritmus a očekávané výstupy. Mezi nejvyužívanější techniky předzpracování textu patří zejména (Miner, 2012):

- **Převod na malá písmena**
  Jak už název napovídá, jedná se o techniku, která pouze převede všechna písmena na malá. Byť se tento krok zdá relativně jednoduchý, jeho dopad je poměrně značný. Pro počítačové algoritmy jsou slova jako například „Kočka“ a „kočka“ dvě různé entity s různými vlastnostmi, i přesto, že představují tu stejnou věc v přirozeném jazyce a nemělo by se s nimi zacházet jinak.
- **Tokenizace**
  Proces tokenizace se zabývá rozdělením souvislého textu na kratší díly. Délka těchto dílů není pevně definována a může se lišit podle aplikace. Nejčastější rozdělení je na jednotlivá slova.
- **Odstranění stop slov**
  V přirozeném jazyce se často objevují slova, která jsou pro lidskou komunikaci důležitá, ale nepřidávají žádný význam. Tato slova se nazývají tzv. stop slova a pro mnoho aplikací NLP jsou nadbytečná. Příkladem stop slov může být například: a, byl, do atd.
- **Stematizace**
  Proces stematizace se zabývá normalizací jednotlivých slov na jejich kořeny za použití algoritmu pro hledání kořenu a odstraňování předpon a přípon. Stematizace nepoužívá žádný slovník a je tedy možné, že vrátí slova, která nejsou součástí zkoumaného jazyka.
- **Lemmatizace**
  Již zmíněný problém stematizace se proces lemmatizace snaží řešit tak, že využívá extensivní slovník pro vyhledání gramaticky korektního základu slova – lemmatu. Obecně je pro získání lemmatu nutno znát více informací, jako již zmíněný slovník nebo kontextu ostatních slov pro určení slovního druhu (Part-of-Speech tagu) zkoumaného slova.

Jak už bylo řečeno, výběr předzpracovacích technik je nesmírně důležitý pro následující zpracování a existují situace, kde využití nesprávné techniky způsobí negativní efekty jako například snížení přesnosti algoritmu nebo ztrátu kontextu věty.

2.3 Strojové učení

Oblast zpracování přirozeného jazyka je s disciplínou strojového učení úzce spojena. Při mnoha aplikacích NLP se využívá různých technik strojového učení pro předpřipravení dat a získání významného výstupu. Obecně se strojové učení zaměřuje na natrénování modelu na vstupních datech, který je poté schopen vytvářet rozhodnutí a predikce nad novými daty na základě znalostí získaných z procesu trénování. Klasické příklady jsou detekce spamu, určení sentimentu nebo vyhledání témat v textu (Bengfort et al., 2018).

Obecně můžeme rozdělit aplikace strojového učení do několika kategorií, které se liší ve formě externího zásahu při učení. Externí zásah může být předem známá informace nebo odměna za správnou odpověď při učení. Hlavní dvě kategorie jsou učení s učitelem a bez učitele.

- **Učení s učitelem (supervised learning)**
  Algoritmy tohoto typu používají předem známou informaci, která se využívá při naučení algoritmu jako požadovaný výstup. Cílem je naučit funkci, která blízce imituje vztah mezi vstupními daty a známou informací tak, že je schopná spolehlivě určit výstup bez předem známé informace (Sammut a Webb, 2010). Typickou aplikací tohoto postupu je tzv. kategorizace neboli přiřazení kategorie danému dokumentu, kterou se tato práce zabývá.
- **Učení bez učitele (unsupervised learning)**
  Na rozdíl od předchozí kategorie učení bez učitele nevyužívá žádný externí zásah. Nemáme žádné předem známé informace a od algoritmu očekáváme, že nalezne smysluplné vazby mezi vstupními daty. Cílem tedy je odhalit strukturu uvnitř vstupních dat. Typickou aplikací tohoto postupu je shlukování, kde se snažíme rozdělit data do několika smysluplných shluků, což může být využito například pro hledání témat v textu (Sammut a Webb, 2010).

2.4 Strukturová reprezentace textových dat

Pro jakékoliv využití algoritmů strojového učení je nejdříve nutné mít textová data převedena do formy, které počítač rozumí. Textová data jsou nestrukturovaná, což znamená, že dokumenty jako takové není možné zcela reprezentovat v číselném či maticovém formátu. Tento fakt pro nás představuje zásadní problém, který nemá žádné universální řešení. Obecně se tento problém řeší výběrem charakteristických rysů dokumentu, které už je možné reprezentovat strukturovaným formátem (Feldman a Sanger, 2007). Samotná problematika strukturované reprezentace textových dat se v posledním desetiletí značně změnila s příchodem deep learning algoritmů a hustých vektorů zachycující kontext slov (Pilehvar a Camacho-Collados, 2020).

Obecně je nutno textová data převést do vektorové podoby. Tyto vektorové reprezentace textů lze rozdělit do řídkých a hustých vektorů. Hlavní rozdíl mezi zmíněnými reprezentacemi je velikost výsledného vektoru. Řídké vektory mají často

tisíce nebo desetitisíce dimenzí, kde je většina dimenzí rovna nule a pouze malá část vektoru má smysluplné hodnoty. Samotný počet dimenzí se odvíjí od vstupních dat a implementace, například můžeme počítat každé unikátní slovo v korpusu jako jednu dimenzi (Feldman a Sanger, 2007).

Toto samozřejmě přináší vysoké nároky na rychlost a paměťový prostor, a proto se v dnešní době zaměřuje spíše na reprezentace hustými vektory. Cílem těchto reprezentací je vytvořit vektor o velikosti v řádu stovek dimenzí, ve kterém každá dimenze nese významnou hodnotu. Další unikátní výhodou těchto vektorů je, že vektory slov využívaných v podobném kontextu směřují do podobného směru, což umožnuje aplikaci různých měr podobnosti (Ganegedara, 2018). V následujících dvou podkapitolách budou popsány často využívané algoritmy reprezentace obou typů.

2.4.1 Řídké vektory (sparse vectors)

Hlavním zástupem této kategorie reprezentace textu je model Bag-of-Words (BoW). Jedná se o jednu z nejstarších metod pro strukturovanou reprezentaci textu. Princip spočívá v zachycení přítomnosti slov z korpusu v dané větě bez ohledu na jejich provázání či slovosled. Výsledkem tohoto procesu vznikne z textového dokumentu vektor o N dimenzích, kde N znamená počet unikátních slov v korpusu. Hlavní myšlenka tohoto postupu je, že dokumenty si jsou podobné, pokud obsahují podobná slova.

Základní BoW model přiřazuje stejný význam pro všechna slova v dokumentu, nicméně je možné využít různých metod pro vážení důležitosti slov. Komplexnější metody vážení slov zahrnují počty výskytů slova na různých místech, jako například v samotném dokumentu, v určité kategorii nebo v celém korpusu. Nejpoužívanější metoda pro vážení slov se nazývá TF-IDF a se skládá ze dvou hlavních složek. První složka *TF* určuje četnost slova v daném dokumentu. Druhá složka *IDF* určuje důležitost slova pro daný dokument pomocí převrácené četnosti v ostatních dokumentech (Feldman a Sanger, 2007).

2.4.2 Husté vektory (dense vectors)

Statické word embeddingy

Jedním z hlavních zástupců této kategorie je model Word2Vec, který využívá neuronové sítě pro získání tzv. word embeddingů, neboli vektorů, které zachycují slovo včetně jeho kontextu. Podobná slova mají podobné vektory a díky této vlastnosti je možné s nimi dělat zajímavé operace ve vektorovém prostoru. Například odečtení vektoru slova Man od vektoru slova King a následné přičtení vektoru slova Woman vrátí vektor blízko vektoru slova Queen. Word2Vec má dva hlavní architektury, a to CBOW (continuous bag-of-words), která nevyužívá pořadí slov v dokumentu, a architekturu Skip-gram, která využívá pořadí slov v dokumentu (Mikolov, 2013). Další významní zástupci této kategorie jsou například FastText nebo GloVe.

Nevýhodou statických word embeddingů získaných například pomocí Word2Vec je, že slova jsou zachována v jednom vše zahrnujícím kontextu. To znamená, že pokud má slovo více významů na základě kontextu, tak se všechny sloučí do jednoho celku, a získané embeddingy mohou v některých případech přinášet nežádoucí výsledky (Liu et al., 2020).

Kontextové word embeddingy

Modely produkující tzv. kontextové word embeddingy se zabývají řešením problému zachování několika významů slova na základě kontextu předešlých reprezentací (Liu et al., 2020).

První implementace kontextových word embeddingů se objevuje v roce 2018 od Peters et al. a značně mění oblast NLP. Navrhovaný model ELMo, neboli Embeddings from Language Models, zcela řeší problém mnoha významů slov pomocí dynamické konstrukce hlubokých embeddingů za běhu. Pro tuto konstrukci se využívá hluboký oboustranný LSTM model pro natrénování dvou jazykových modelů, jeden zachycující levý kontext slova a druhý zachycující pravý kontext slova.

V původním článku Peters et al. (2018) popisují hluboké embeddingy získané pomocí ELMo v tom smyslu, že reprezentují kombinaci všech interních stavů natrénovaných jazykových modelů pro dané slovo, což produkuje vysoce kvalitní reprezentace slov. Vlastnost kombinace vnitřních stavů má další výhody, Peters et al. (2018) ukazují, že stavy na vyšších úrovních zachycují aspekty kontextu, v kterém se dané slovo aktuálně nachází. Naopak stavy na nižších úrovních modelují syntaktické aspekty slova. Tyto vlastnosti umožnují ELMo embeddingy využít pro velké množství NLP úloh se state-of-the-art výsledky. Ačkoli ELMo využívá oboustranného LSTM modelu, jeho embeddingy nejsou zcela oboustranné, jelikož se embedding slova získá konkatenací modelu, který se zaměřuje na kontext zleva-do-prava a modelu, který se zaměřuje na kontext zprava-do-leva. Model tedy není schopen zahrnout oba kontexty současně.

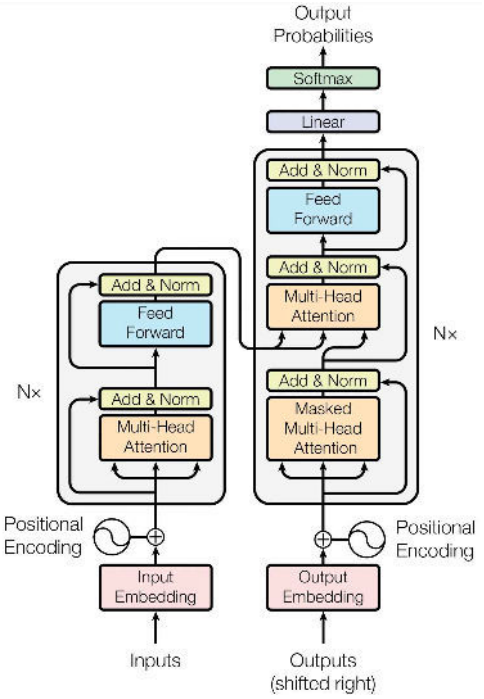
V předešlých přístupech bylo nutné uchovávat slovník výrazů, kterým model rozumí a dokáže zpracovat. V případě modelu ELMo se využívá již zmíněného LSTM modelu pro dynamickou konstrukci embeddingů pro jakýkoliv vstup. Tato vlastnost je umožněna tím, že nepracuje na úrovni jednotlivých slov či sekvencí, ale na úrovni znaků, které tvoří slova a sekvence (Taylor, 2019).

Ačkoli ELMo a podobné přístupy produkují kvalitní výsledky ve srovnání s řídkými reprezentacemi a přetrénovanými embeddingy typu Word2Vec, existují nedostatky, které mohou aplikaci těchto přístupů omezit. Hlavní problém je obtížná paralelizace LSTM, což způsobuje pomalé trénovaní a inferenci (Hashesh, 2020).

Tyto problémy se model nazývaný transformer, založený na tzv. self-attention algoritmu, snaží vyřešit a bude dále popsán v následující kapitole 2.5.

2.5 Transformer

Model Transformer, navržen Vaswani et al. (2017), značně změnil oblast zpracování přirozeného jazyka od svého vzniku. Klíčovou vlastností modelu je využití tzv. self–attention mechanismu neboli mechanismu interní pozornosti. Tento mechanismus se stará o zaměření na určitá slova ve vstupní větě při vytváření vektorové reprezentace každého slova věty. Model je tvořen dvěma hlavními části, a to: kodér (encoder) a dekodér (decoder). V původním návrhu se kodér stará o převod sekvence symbolů (slova, věty) na kontinuální sekvenci (vektor), která je předána dekodéru spolu s dalšími vstupy. Před vstupem do kodéru jsou získány embeddingy slov například pomocí Word2Vec. Dekodér poté vstupy transformuje a vrací opět sekvenci symbolů, která se využije jako vstup pro jeho další krok (Vaswani et al., 2017). Obecná architektura transformer modelu je vyobrazena na obrázku 1.



Obr. 1: Architektura transformeru, kodér (vlevo), dekodér (vpravo)
Zdroj: Vaswani et al., 2017

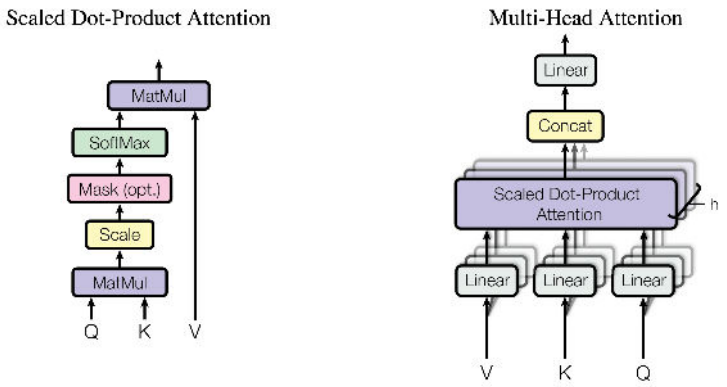
V praxi není nutné se striktně řídit navrhovanou architekturou a mnohé implementace zmíněných mechanismů využívají pouze části obecného transformer modelu. Existují tři hlavní kategorie, do kterých implementace spadají, a to: modely pouze s kodérem, modely pouze s dekodérem a kodér dekodér modely. Každá varianta má určité výhody, které je vhodné využít pro specifické úlohy. Například modely pouze s kodérem jsou vhodné pro úlohy vyžadující porozumění vstupu jako například klasifikace vět nebo rozpoznávání pojmenovaných entit. Modely pouze s dekodérem jsou naopak vhodnější pro úlohy typu generování textu. Kodér dekodér modely, podobně jako modely pouze s dekodérem, jsou vhodné pro generování textu, nicméně

je možné je využít i pro situace, kde je vyžadován vstup, jako například překlad nebo sumarizace (Hugging Face, 2022).

Mnohé aplikace zpracování přirozeného jazyka dosahující state-of-the-art výsledků využívají modely založené na transformeru, a proto budou značným zaměřením této práce.

Jak už bylo zmíněno, klíčová vlastnost transformer modelu je využití self–attention mechanismu. Princip fungování mechanismu spočívá ve využití ostatních pozic ve vstupu (ostatních slov) při zpracovávání aktuálního slova, což umožňuje získání kvalitnější reprezentace a porozumění využití slova v daném případě. Při zpracování vstupní sekvence se nejdříve pro každé slovo tvoří tři vektory: query (Q), key (K) a value (V) tak, že se embedding vstupního slova vynásobí váhovými maticemi W^Q, W^K a W^V získané z trénovacího procesu. Poté se pro každé slovo vypočítají skóre všech slov ve vstupní posloupnosti. Tato skóre uvádí, jak je dané slovo v sekvenci důležité pro konkrétně analyzované slovo s tím, že nejvyšší důležitost má obvykle konkrétně analyzované slovo samo. Získané skóre se následně podělí $\sqrt{d_V}$, tedy odmocninou počtu dimenzí vektoru V a použije se softmax funkce pro odstranění kontextu nerelevantních slov. Na závěr se skóre sečtou, což je výsledek procesu pro konkrétní analyzované slovo. V praxi se tyto operace řeší maticově pro celý vstup, což umožnuje modelu efektivně zpracovat několik dokumentů najednou (Alammar, 2018). Tento proces je nazýván Scaled Dot-Product Attention a jeho struktura je vyobrazena na obrázku 2 vlevo.

Proces získání Scaled Dot-Product Attention je v modelu transformer součásti tzv. Multi-Head Attention, což dále umožnuje rozšířit zachycení kontextu daného slova, jelikož self–attention slova často zdůrazňuje sama sebe. Dále Multi-Head Attention umožňuje existenci několika reprezentačních podprostorů, které se mohou zaměřovat na různé aspekty slova. Princip je postaven na skládání několika Scaled Dot-Product Attention hlav za sebou (v původním návrhu celkem 8 hlav), kde se v každé využívá jiné matice W^Q, W^K a W^V. Tyto matice jsou na začátku trénování náhodně inicializovány a průběhem trénování se zaměří na jiný aspekt slov. Proces uvnitř každé hlavy je identický již popisovanému, nicméně naráží se na problém eventuálního využití několika výstupů pro dopřednou neuronovou síť (Feed Forward), která očekává pouze jeden vstup. Pro vyřešení tohoto problému se na závěr operace konkatenují výstupy všech hlav a vynásobí se váhovou maticí W^O, která byla trénována společně s modelem (Alammar, 2018). Diagram Multi-Head Attention v transformer modelu je vyobrazen na obrázku 2 vpravo.



Obr. 2: Scaled Dot-Product Attention (vlevo), Multi-Head Attention (vpravo)
Zdroj: Vaswani et al., 2017

2.5.1 BERT

Jak už bylo zmíněno, model transformer značně změnil oblast zpracování přirozeného jazyka, nicméně široké použití zažily až jeho adaptace. Nejznámější model založený na transformeru se nazývá BERT, zkráceno z Bidirectional Encoder Representations from Transformers. BERT využívá architekturu založenou na kodérech transformeru, které jsou postavené za sebou v několika vrstvách (Rogers et al., 2020).

Na rozdíl od původního transformer modelu BERT nepotřebuje pro svoji inicializaci předtrénované embeddingy slov, a místo toho si dynamicky tvoří své reprezentace slov na základě kontextu jejich výskytu. Nicméně konstatovat, že BERT pracuje se slovy, je poněkud zavádějící, jelikož pro tokenizaci vstupních dokumentů využívá tzv. WordPiece algoritmus. WordPiece využívá vnitřní slovník obsahující okolo 30 000 nejčastěji vyskytujících se tokenů v anglickém jazyce. Slovník se dále generuje a adaptuje pro vstupní data zároveň s modelem při trénování. Princip spočívá v rozdělení vstupu na části obsažené v předtrénovaném slovníku. Tyto části mohou mít formu celých slov, samotných znaků nebo pod částí slov jako například předpony a přípony. Výhodou tohoto přístupu je, že je model schopen rekonstruovat dobré reprezentace slov, které ve slovníku nejsou na základě jeho pod částí (McCormick, 2019). Příklad tokenizace slova, které není ve slovníku, část ## signalizuje, že se jedná o navazující token:

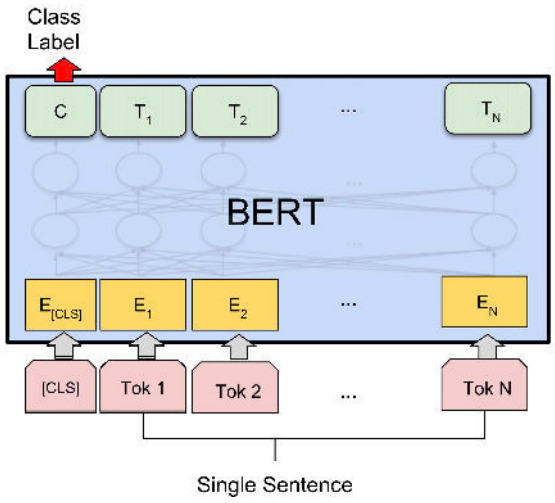
$$embeddings \rightarrow [em, \#\#bed, \#\#ding, \#\#s]$$

BERT dokáže pracovat maximálně se vstupem, který lze tokenizovat do 512 tokenů. Při využívání vstupu, který je delší než 512, se typicky tokenizované dokumenty zkrátí na 512 a nadbytečné tokeny se vypustí. Toto samozřejmě znamená, že se vypustí i jakýkoliv význam a informace po 512 tokenech. BERT dále očekává speciální tokeny [CLS] na začátku dokumentu a [SEP] na konci dokumentu. Při aplikaci BERTa na kratší dokumenty je možné limitovat maximální počet tokenů na méně než 512, což výrazně zrychluje trénování a inferenci (Briggs, 2021a).

Unikátní vlastnost modelu BERT oproti předchozím state-of-the-art přístupů je, že dosahuje plné obousměrnosti při získávání reprezentace dokumentů. Ve srovnání s modely na bázi transformeru OpenAI GPT a GPT-2, které využívají trénování jazykového modelu pomocí metody zleva-do-zprava, kde se využívá pro získání reprezentace slova pouze jeho levý kontext, BERT používá trénování pomocí metody maskování slov (MLM) a predikce následující věty (NSP) pro získání reprezentace slov s využitím levého i pravého kontextu slova současně. Tímto se také liší od modelu ELMo. Ačkoli ELMo používá pravý i levý kontext slova, neprovádí výpočet současně a využívá dva jazykové modely, kde se jeden zaměří na směr zleva-do-prava a druhý na směr zprava-do-leva a jejich výstupy pro kontextovou reprezentaci tokenu se poté konkatenují. Výsledek dosažení plné obousměrnosti BERTa se projevuje v hlubokém porozumění lidského jazyka jako takového (Devlin et al. 2018).

Jak už bylo zmíněno, BERT používá pro své trénování dvě různé metody, nicméně než budou představeny, je nutné porozumět, jakým způsobem je možné trénovat BERT. Trénování obecně probíhá ve dvou fázích, a to: předtrénování (pre-training) a doladění (fine-tuning). Při předtrénování se model učí na neoznačených datech již zmíněnými technikami, kde je cílem naučit model obecné rysy jazyka. Při doladění se poté již předtrénovaný model trénuje na konkrétním úkolu, jako například klasifikace dokumentu nebo rozpoznání pojmenovaných entit v textu. Výhodou tohoto přístupu je snadné využití společného základu pro řešení různých úloh (Devlin et al., 2018).

Při aplikaci modelu BERT pro klasifikaci se využívá speciálního tokenu [CLS]. Jedná se o speciální klasifikační token, který se model naučí už při předtrénování. Výstupný embedding tokenu [CLS] se poté použije jako vstup pro klasifikační vrstvu (Devlin et al., 2018). Diagram využití modelu BERT pro klasifikaci věty či dokumentu lze vidět na obrázku 3.



Obr. 3: Diagram využití modelu BERT pro klasifikaci
Zdroj: Devlin et al., 2018

Výpočetní nároky

BERT má ve svém původním návrhu dvě hlavní verze, a to: $\text{BERT}_{\text{BASE}}$ a $\text{BERT}_{\text{LARGE}}$. Hlavní rozdíl je v množství trénovatelných parametrů, které se odvíjí od množství kodérů a hlav v multi-head attention mechanismu. $\text{BERT}_{\text{BASE}}$ obsahuje celkem 12 kodérů za sebou, 12 multi-head attention hlav a počet dimenzí skrytých stavů rovno 768. $\text{BERT}_{\text{LARGE}}$ disponuje mnohem větším množstvím trénovatelných parametrů s 24 kodéry, 16 multi-head attention hlavy a velikost vnitřních stavů rovno 1024. Ve srovnání to je celkem 110 milionů trénovatelných parametrů pro $\text{BERT}_{\text{BASE}}$ a 340 milionů pro $\text{BERT}_{\text{LARGE}}$. Vyšší množství parametrů může přinášet lepší výsledky na konečném úkolu, nicméně je nutné počítat s mnohem vyššími požadavky na výpočetní výkon jak pro samotné doladění, tak pro inferenci. Obě varianty modelu byly předtrénovány na datech skládajících se z BooksCorpus[1] a anglické Wikipedie, obsahující celkem přes 33 milionů slov. Předtrénování $\text{BERT}_{\text{LARGE}}$ vyžadovalo 4násobný výpočetní výkon oproti $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018).

Jelikož BERT využívá architekturu transformeru jako svůj základ, obsahuje několik stovek milionů trénovatelných parametrů. Trénování všech parametrů z nuly vyžaduje obrovské množství výpočetního výkonu, i pro velikost $\text{BERT}_{\text{BASE}}$, a proto je využita strategie předtrénování kvalitního základu a následné doladění na potřebné úlohy. Proces doladění je velice intuitivní a spočívá v přidání nové hlavy nad kodéry BERTa, která řeší požadovaný problém, jako například klasifikační hlava nebo hlava řešící úkol otázka a odpověď. Dále je možné trénovat parametry pouze těchto přidaných hlav, nebo trénovat všechny parametry modelu. Výhodou trénování všech parametrů je lepší přizpůsobení modelu na konkrétní doménu, nicméně je nutné počítat s mnohem vyššími požadavky na výpočetní výkon (Briggs, 2021b).

Zmíněné nároky na výpočetní výkon mohou být často prohibitivní pro využití v produkčních aplikacích, a proto se po prvním návrhu modelu BERT vyskytlo několik způsobů, jak model zrychlit. Jednou z hlavních metod zrychlení je tzv. destilace. Jedná se o metodu komprese modelu, která převádí znalosti velkého učitelského modelu na menší studentský model, který je trénován tak, aby napodoboval výstupy učitele (Sapunov, 2019). Nejznámější destilace modelu BERT se nazývá DistilBERT. Oproti $\text{BERT}_{\text{BASE}}$ obsahuje 40 % méně trénovatelných parametrů při zachování 97 % schopnosti rozumění jazyka. Navíc má o 60 % rychlejší inferenci (Sanh et al., 2019).

Rozšíření modelu BERT

Od svého vzniku BERT dosahuje state-of-the-art výsledků ve velkém množství úloh, což vede k mnoha pracím adaptujícím a derivujícím jeho princip. Existuje několik prací rozšiřující BERT na jiné jazyky než pouze angličtinu, například Arkhipov et al. (2019) rozšiřují vícejazyčnou variantu modelu o komplexní znalosti slovanských jazyků, a značně tak zvyšují kvalitu výsledků pro všechny slovanské jazyky oproti obecné vícejazyčné variantě. Výsledný model se nazývá Slavic BERT a slouží jako jeden z nejlepších modelů založených na transformerech pro český jazyk.

[1] Dataset 11 038 vybraných knih (Zhu et al., 2015)

Další práce se zabývají procesem trénování a jeho optimalizací. Liu et al. (2019) se zaměřují na optimalizaci předtrénování BERTa, kde používají 10krát větší množství dat (z 16 GB na 160 GB) a pouze metodu MLM, kde dynamicky mění maskovaná slova pro každou trénovací epochu. Výsledný model RoBERTa poskytuje mírně lepší výsledky v prováděných testech než $\text{BERT}_{\text{LARGE}}$.

Reimers a Gurevych (2019) (Sentence-BERT) se zaměřují na adaptaci BERTa pro získání embeddingů celých vět nebo dokumentů, které poté lze porovnávat pomocí kosinové podobnosti. Tohoto dosahují s využitím siamské neuronové sítě, která se skládá ze dvou identických sítí, což umožnuje zpracovat dva dokumenty současně. Získaný model byl testován na úloze zjištění nejpodobnějšího páru dokumentů, kde Sentence-BERT model dosáhnul obrovského zrychlení oproti výchozímu BERT modelu. Při hledání nejpodobnějšího páru v kolekci 10 000 dokumentů výchozí BERT dospěl k výsledku za 65 hodin, kdežto Sentence-BERT model dosáhl výsledku za 5 sekund. Tato derivace modelu BERT může být vhodná například pro zjištění, zda je zákaznický požadavek duplicitní již existujícímu.

2.6 Kategorizace zákaznických požadavků

Problematika kategorizace neboli klasifikace je jedna z nejstarších a nejpopulárnějších využití NLP. Aplikace kategorizace pro data typu zákaznických požadavků může mít několik forem. Nejčastější forma je kategorizace typu požadavku, tedy zda se jedná například o reklamaci, dotaz, stížnost či ke které kategorii zboží požadavek patří. Dále je možné pomocí kategorizace zjistit, zda je požadavek duplikát už existujícího, což může drasticky snížit čas odezvy operátora nebo duplicitní požadavek nemusí vyžadovat žádný zásah operátora. Další možnost kategorizace je přiřazení vhodného operátora na základě vlastností požadavku, například pomocí samotného textu nebo kombinací několika charakteristik. Existuje opravdu mnoho aplikací kategorizace pro data typu zákaznických požadavků, a proto je nutné zvolit vhodné metody pro vybranou aplikaci.

V následujících kapitolách budou popsány metody pro vyhodnocení výsledků klasifikace a srovnáno několik existujících řešení problematiky kategorizace zákaznických požadavků.

2.6.1 Vyhodnocení kategorizace

Pro vyhodnocení výsledků klasifikace je nutné stanovit postupy a metriky, podle kterých je možno jednoznačně porovnat a hodnotit jednotlivé přístupy a algoritmy. Typické postupy zahrnují rozdělení vstupního datasetu do dvou hlavních částí – trénovací a testovací. Trénovací dataset se využívá pro natrénovaní klasifikátoru a bývá okolo 80 % původního datasetu. Testovací dataset není využitý v rámci trénovaní a využívá se pro ověření schopnosti generalizace znalostí již natrénovaného klasifikátoru na předem neviděných datech. Hlavní podmínka pro testovací dataset

je, že jeho instance musí mít podobný charakter jako instance trénovacích dat (Žižka et al., 2020).

V některých případech se dataset rozdělí na tři části – trénovací, validační a testovací, kde evaluační dataset slouží pro výběr modelu s nejlepším výkonem při optimalizaci parametrů klasifikátoru. Typicky se nejdříve dataset rozdělí na části trénovací a testovací a poté se z trénovací části dále rozdělí na trénovací a validační. Další možnost postupu rozdělení datasetu je pomocí křížové validace neboli cross validation. Princip křížové validace spočívá v rozdělení datasetu do q stejně velkých segmentů, kde je jeden segment využitý pro testování a ostatní $q-1$ pro trénování. Tento proces se poté provede *q-krát*, kde se v každé iteraci využije jiný segment jako testovací (Aggarwal, 2018).

Samotné metriky pro posouzení výkonu klasifikátoru využívají čtyři hlavní komponenty pro svůj výpočet. Jedná se o True Positive (TP), True Negative (TN), False Positive (FP) a False Negative (FN) (Žižka et al., 2020).

- **True Positive (TP)**
  Případ, kdy instance (dokument) patří do kategorie C a klasifikátor jej korektně přiřadil do kategorie C.
- **True Negative (TN)**
  Případ, kdy instance (dokument) nepatří do kategorie C a klasifikátor jej korektně nepřiřadil do kategorie C.
- **False Positive (FP)**
  Případ, kdy instance (dokument) nepatří do kategorie C a klasifikátor jej nesprávně přiřadil do kategorie C.
- **False Negative (FN)**
  Případ, kdy instance (dokument) patří do kategorie C a klasifikátor jej nesprávně přiřadil do kategorie jiné než C.

Matice záměn

Často také používána v anglickém názvu confusion matrix, matice záměn slouží pro jasné shrnutí a vizualizaci všech čtyř hlavních komponent pro vypočítání metrik. Poskytuje jasný přehled o chybách, kterých se klasifikátor dopustil, ale především lze snadno odhalit, jaké typy chyb se vyskytují, neboli v kterých oblastech má klasifikátor největší problémy (Raschka, 2015).

Matice záměn se často definuje pro binární klasifikační problémy, vyobrazeno v tabulce 1. V tabulce 2 je vyobrazena možnost aplikace i pro problémy s několika kategoriemi tak, že se každá kategorie rozdělí na binární problém, kde je pozitivní kategorie aktuálně zkoumaná kategorie (v tabulce kategorie „Stížnost“) a negativní kategorie jsou všechny ostatní.

Tab. 1: Matice záměn pro binární klasifikační problém

		Předpovězená kategorie	
		Pozitivní	Negativní
Skutečná kategorie	Pozitivní	TP	FP
	Negativní	FN	TN

Tab. 2: Matice záměn pro klasifikační problém s třemi kategoriemi z hlediska kategorie Stížnost

		Předpovězená kategorie		
		Reklamace	Stížnost	Dotaz
Skutečná kategorie	Reklamace	TN	FP	TN
	Stížnost	FN	TP	FN
	Dotaz	TN	FP	TN

Metriky pro vyhodnocení klasifikátoru

Všechny následující metriky jsou definovány pro binární klasifikační problémy, tedy případy, kde se pracuje pouze s dvěma kategoriemi. U případů s více kategoriemi, jako například kategorizace zákaznických požadavků, je nutné metriky počítat pro každou kategorii zvlášť a poté využít některou z metod pro sdružení jednotlivých výsledků. Existují dva hlavní postupy sdružení metrik, a to makro průměr a mikro průměr (Sokolova a Lapalme, 2009).

Makro průměr spočívá ve vypočítání aritmetického průměru metrik všech kategorií, což znamená, že zachází se všemi kategoriemi stejně. Mikro průměr využívá součet všech potřebných komponent (TP, TN, FP, FN) všech kategorií pro výpočet průměru metriky, což může být vhodné pro případy, kdy je dataset vyvážený (Sokolova a Lapalme, 2009). Existuje také sdružení váženým průměrem, implementováno v rámci knihovny Scikit-Learn, což spočívá ve výpočtu váženého průměru metrik kategorií podle počtu instancí v dané kategorii. Tato vlastnost může být vhodná pro nevyvážené datasety nebo pro datasety, kde jsou kategorie s méně dokumenty méně významné než kategorie s více dokumenty (Scikit-Learn, 2022).

Accuracy

Accuracy je jednou z nejzákladnějších metrik pro vyhodnocení klasifikace. Princip spočívá v podílu správně klasifikovaných instancí na celkovém počtu zkoumaných instancí. Tato metrika může být vhodná pro případy, kdy je dataset dobře vyvážený, nicméně rychle se můžeme dostat k zavádějícím výsledkům při používání nevyváženého datasetu, kde má některá z kategorií mnohem větší zastoupení (Gu et al., 2009).

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

Precision

Metrika precision se zaměřuje na zkoumání klasifikovaných instancí do kategorie C tak, že odpovídá na otázku: jaký podíl klasifikovaných dokumentů do kategorie C opravdu patří do kategorie C (Gu et al., 2009).

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

Recall

Metrika Recall se zaměřuje na zkoumání instancí, které opravdu patří do kategorie C tak, že odpovídá na otázku: jaký podíl dokumentů patřící do C byl korektně klasifikován jako patřící do C (Gu et al., 2009).

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

F1 skóre

Obecně je žádoucí, aby klasifikátor měl hodnoty Precision i Recall co nejvyšší současně. F1 skóre se snaží obě metriky vyvážit a zároveň poskytnout jednu všezahrnující hodnotu, díky které je možno snadno srovnávat různé klasifikátory. Funguje na principu harmonického průměru Precision a Recall, což mezi dílčimi metrikami udržuje rovnováhu (Gu et al., 2009).

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

2.6.2 Tradiční přístupy

Byť oblast zpracování přirozeného jazyka zažila značné průlomy v posledních letech, mnoho prací a aplikací stále využívá osvědčené tradiční metody pro aplikaci strojového učení na textová data. Konkrétně pro oblast automatické kategorizace zákaznických požadavků lze najít několik prací z posledních let využívajících řídké reprezentace textů s minimálním využitím moderních metod umělé inteligence a neuronových sítí.

Příkladem může být práce od Parmar et al. (2018), kde bylo testováno několik metod pro vytvoření klasifikátoru pro zprávy zákaznické podpory. Jako základ zde autoři využili řídkou reprezentaci dokumentů pomocí TF-IDF s menšími předzpracovacími kroky obsahující zejména čištění datasetu od prázdných hodnot. Autoři využívali velice nevyvážený dataset obsahující tisíce dokumentů s dvanácti odlišnými kategoriemi, na kterém zkoušeli celkem pět různých klasifikátorů a to: Multinominal Naïve Bayes (MNB), Support Vector Machine (SVM), Decision Tree, Random Forest a K-Nearest Neighbors (KNN). Nejlepší výsledky pro zvolenou metriku accuracy dával klasifikátor SVM s 63,03 %. Konečný výkon klasifikátoru této práce je relativně nízký s vysokou mírou chybovosti a dalo by se pochybovat o jeho užitečnosti. Zároveň využití metriky accuracy pro nevyvážený dataset s velkým množstvím kategorií není ideální z důvodů vysvětlených v předchozí kapitole 2.6.1.

Nicméně slabé výsledky jedné práce nemusí nutně znamenat, že využité metody nejsou vhodné pro daný problém, jak je možno pozorovat u práce Paramesh et al. (2018), kde je využito několik stejných přístupů jako u práce Parmar et al. (2018). Paramesh et al. (2018) opět využívali řídkou reprezentaci dokumentů pomocí TF-IDF, nicméně navíc využili chí kvadrátový test pro další filtrování slov s malým významem. Práce se značně liší od předešlého pokusu množštvím předzpracovacích kroků. Autoři se zaměřují na důkladné očištění vstupních dat od nežádoucích jmen, emailových adres, telefonních čísel atd. Následně také odstraňují stop slova a využívají převzorkovacích a podvzorkovacích metod pro vyvážení datasetu, který obsahuje okolo 10 000 dokumentů s 18 odlišnými kategoriemi. Samotné použité klasifikační algoritmy jsou velmi podobné jako v předem popisované práci, nicméně kde se Paramesh et al. (2018) opět značně liší, je ve využití metod pro kombinaci několika natrénovaných klasifikátorů pro zlepšení kvality predikce. Zkoumané metody zahrnují Bagging, Boosting a Voting Ensemble, kde nejlepší výsledky dosáhli pomocí Bagged Decision Tree klasifikátoru se zvolenou metrikou accuracy 92,04 %.

V druhé zkoumané práci se autorům osvědčily techniky kombinování několika klasifikátorů pro dosažení vyšší accuracy, než by bylo možné pouze s jedním klasifikátorem. Technika *bagging*, která autorům poskytovala nejlepší výsledky, funguje na principu rozdělení vstupního datasetu do několika částečně nezávislých vzorků, které se následně využijí pro souběžné natrénování několika slabých klasifikátorů. Výkon těchto klasifikátorů bývá mírně lepší než náhodné odhady, nicméně při jejich kombinaci, buď hlasovacím systémem nebo průměrem predikcí, mohou dosahovat dobrých výsledků. Autoři dále zkoumali metodu *boosting*, která jim poskytovala mnohem horší výsledky než bagging. Jedná se o další metodu komise klasifikátorů, která trénuje slabé klasifikátory sekvenčně (Zhou, 2012). Poslední zkoumaná metoda – *Voting Ensemble* funguje na principu sčítání predikcí několika modelů. Výhodou Voting Ensemble kombinace je, že je možné využít různé klasifikační algoritmy nebo algoritmy s různými trénovacími parametry (Brownlee, 2020).

Další práce řešící problematiku kategorizace tiketů ukazuje konečný dopad na finanční stránku podniku po aplikaci řešení pomocí strojového učení. Eichhorn (2020)

opět využívá tradiční metody pro realizaci cíle s využitím řídké reprezentace dokumentů pomocí TF-IDF. Preprocessing dat se skládal zejména z lemmatizace, odstranění stop slov a interpunkce. Místo, kde se tato práce liší od předchozích řešení, je ve filtraci řídce vyskytujících se kategorií. Využívaný dataset obsahuje téměř 8000 tiketů s 13 odlišnými kategoriemi a autor záměrně odstraňuje kategorie obsahující méně než 100 tiketů, což zmenšuje počet odlišných kategorií z 13 na 8 a značně tak usnadňuje práci klasifikátoru. Dalo by se spekulovat, zda je toto správný přístup, jelikož autor ztrácí schopnost předpovídat méně se vyskytující kategorie, nicméně výhoda tohoto přístupu je kvalitnější zpracování často se vyskytujících kategorií. Opět zde autor využívá několik různých tradičních klasifikačních algoritmů, kde vychází metoda *Logistic Regression* nejlépe z 8 zkoušených klasifikátorů. Cílem práce bylo zlepšit kategorizaci tiketů prováděnou externí firmou s 40% accuracy metrikou, což se autorovi podařilo dosáhnout s accuracy rovno 85 %. Důsledkem zapojení klasifikátoru firma ušetřila téměř $600 000 ročně a výrazně se zrychlila doba odezvy agentů podpory. Autor navíc uvádí, že se uživatelské zkušenosti s firemní podporou výrazně zlepšily.

Z popisovaných prací je evidentní, že tradiční metody a algoritmy pro klasifikaci zákaznických požadavků jsou stále velice relevantní pro vybudování robustního systému. Nicméně záviset na tradičních metodách by mohlo vést k systému, který není na vrcholu svého potenciálu. Proto je vhodné prozkoumat průlomy v oblasti NLP a možnosti jejich aplikace na kategorizaci zákaznických požadavků.

2.6.3 Moderní přístupy

Moderní přístupy oblasti zpracování přirozeného jazyka stále nejsou pro problematiku kategorizace zákaznických požadavků důkladně prozkoumané a lze najít různá řešení využívající kompletně odlišné techniky.

První zkoumaná práce od Zhong a Li (2019) se zaměřuje na kategorizaci přepisů zákaznických hovorů. Autoři zde zkoumají několik různých postupů obecně využívajících textové reprezentace pomocí statických word embeddingů, kde bylo pomocí experimentů zjištěno, že předtrénované GloVe vektory poskytují nejlepší základ pro vybudování systému. Využívaný dataset obsahuje přes 9000 dokumentů a 4 odlišné kategorie. Texty přepisů byly předzpracovány zejména technikami pro očištění textů od nežádoucích slov a výrazů pomocí odstranění stop slov. Klíčová charakteristika této práce je využití konvoluční neuronové sítě (CNN) pro vytvoření klasifikátoru založeného na umělé inteligenci. Autoři dále popisují, že výzva tohoto přístupu spočívala ve využití delších dokumentů jako vstup do CNN, jelikož se CNN tradičně využívá spíše pro kratší texty. Navrhovaný přístup byl otestován na třech různých datasetech, kde vždy dosahoval velmi dobrých výsledků, zejména u hlavního datasetu přepisů zákaznických hovorů, kde vybudovaný klasifikátor dosahoval F1 skóre rovno 93 %.

Všechny doposud srovnávané práce využívaly datasety s počtem dokumentů v řádu tisíců. Tento fakt však neznamená, že tomu tak je vždy. Opuchlich (2019) se

zaměřuje na klasifikaci tiketů ze SAP databáze obsahující několik milionů dokumentů a přes 4000 kategorií spadajících do dvou hlavních oblastí, a to IT a HR. Autor se zaměřuje na srovnání a eventuální kombinaci tradičních a moderních přístupů s využitím řídkých i hustých vektorových reprezentací tiketů, konkrétně TF-IDF a FastText. Práce se dále zaměřuje na dopad různých předzpracovacích technik, jako například odstraňování stop slov, lemmatizace a odstraňování kategorií s malým množstvím tiketů. Jako výsledek analýzy dopadu přezpracovacích technik bylo zjištěno, že odstraňování řídce se vyskytujících kategorií má minimální dopad na konečný výkon klasifikátoru. Co se týče samotné klasifikace dokumentů, tak autor zvolil dvoustupňový klasifikátor, jelikož byl charakter HR a IT tiketů příliš odlišný. Nejdříve se tedy natrénoval binární klasifikátor pro rozdělení tiketů do HR nebo IT oblasti, s accuracy rovno 97,46 %, a poté se natrénovaly oddělené klasifikátory pro každou oblast. Jelikož má využitý dataset opravdu vysoké množství kategorií, tak se autor rozhodl poskytovat 5 nejpravděpodobnějších predikcí klasifikátoru, což významně zvýšilo přesnost souhrnného systému. Na všech úrovních klasifikace vyprodukoval klasifikátor natrénovaný tradičním přístupem mírně lepší výsledky než FastText, nicméně při kombinaci obou pomocí Voting Ensemble bylo dosaženo accuracy 81,4 % pro IT klasifikátor a 78,9 % pro HR klasifikátor, zlepšující výsledky o 2–3 %.

Byť zmíněné dvě práce používají modernější techniky než práce z minulé kapitoly, žádná z nich nevyužívá poslední inovace v oblasti NLP. Pro přehled aplikací využívajících poslední state-of-the-art metody, jako například modely založené na transformeru, je nutné opustit sféru zákaznických požadavků a prozkoumat práce zabývající se daty s podobným charakterem jako tikety helpdesku. Minaee et al. (2021) se ve své práci zabývali srovnáním různých modelů založených na hlubokém učení, kde zkoumali několik nejčastějších využití NLP včetně vícetřídní klasifikace. V experimentu klasifikace tématu dokumentu využívali dataset DBpedia, který obsahuje přes 600 000 dokumentů a 14 odlišných kategorií. Autoři zkoušeli celkem 9 různých modelů, kde všechny dosahovaly velmi dobrého F1 skóre nad 98 %, přičemž nejlepší výsledky poskytoval model $\text{BERT}_{\text{LARGE}}$ s F1 rovno 99,32 % a model XLNET s F1 rovno 99,38 %. Lze zde vidět nadvládu transformeru, jelikož oba nejvýkonnější modely jsou na něm založené. Při úkolu kategorizace 127 000 sumarizací novinových článků do 4 odlišných kategoriích autoři opět vidí podobné trendy, kde transformer modely poskytují nejlepší výsledky. V tomto experimentu dosahuje model XLNET F1 rovno 95,51 %, což je zhruba o 3 % lepší než modely využívající CNN.

Další práce využívající model BERT se zaměřuje na možnosti využití aktivního učení při vícetřídní klasifikaci. Jedná se o formu strojového učení, kde se využívají neoznačené dokumenty. Učicí algoritmus vybere z datasetu podmnožinu, která se poté označí. Cílem je naučit kvalitní algoritmus s využitím co nejméně označených dokumentů (Sammut a Webb, 2010). Prabhu et al. (2021) využívali dataset krátkých popisků transakcí o velikosti 55 000 dokumentů s 10 odlišnými kategoriemi. Pro vektorizaci a klasifikaci využili model DistilBERT, kde poté zkoumali různé algoritmy aktivního učení. Autoři zjistili, že s využitím aktivního učení je možné dosáhnout

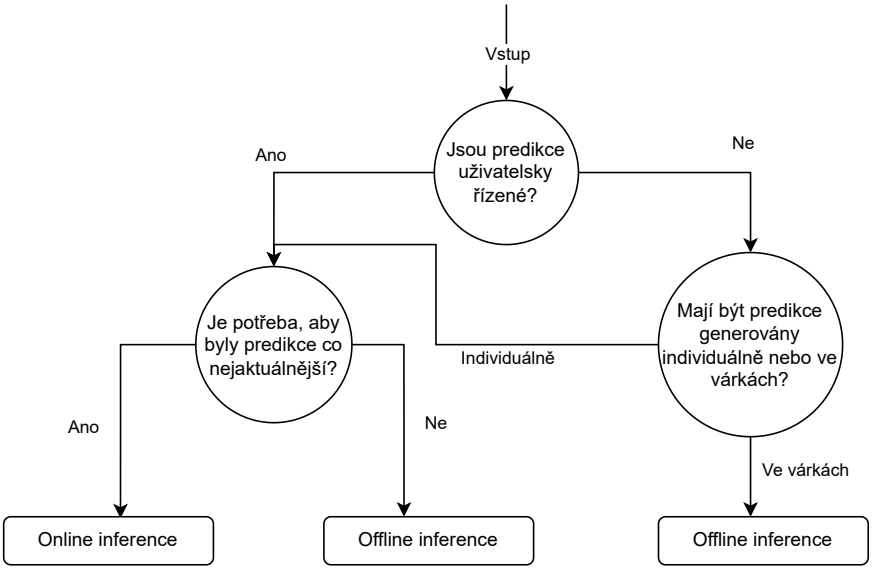
stejného F1 skóre rovno 83 % jako při využívání klasického učení s učitelem, a navíc lze dosáhnout redukci potřebných označených dokumentů o 85 %.

2.7 Aplikace algoritmů

Pro vybudování automatizovaného systému pro kategorizaci zákaznických požadavků je nejen potřeba vyvinout samotný model strojového učení, ale i navrhnout vhodnou strukturu nasazení tohoto modelu. V následujících podkapitolách budou popsány možnosti nasazení modelu pro vybudování automatizovaného systému pro kategorizaci.

2.7.1 Typy aplikací

V oblasti aplikace a nasazení modelů strojového učení obecně existují dva hlavní přístupy, a to: online inference, neboli inference v reálném čase, a offline inference, neboli várková inference. Hlavní rozdíl spočívá ve způsobu získání predikce. Online inference se snaží poskytnout co nejaktuálnější výsledky v co nejrychlejším čase pro jeden konkrétní dokument, kdežto offline inference se snaží poskytnout efektivní předpovědi pro větší množství dokumentů. Který přístup se využije,záleží na daném řešeném problému a formě požadovaného systému. Obecně je vhodné odpovědět na otázky typu: jak často mají být predikce generovány? Jaká je požadovaná doba odezvy? Jsou predikce uživatelsky řízené (Microsoft, 2021)? V obrázku 4 je popsán způsob zvolení typu nasazení ve formě rozhodovacího stromu podle Microsoft Azure dokumentace.

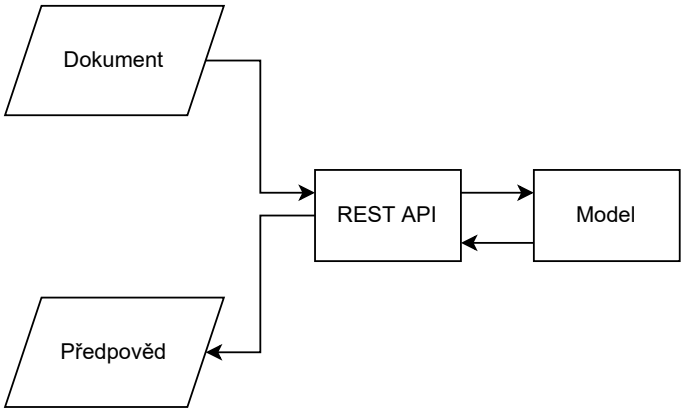


Obr. 4: Rozhodovací strom pro zvolení typu nasazení
Zdroj: Microsoft, 2021

Online inference

Inference v reálném čase se obvykle zaměřuje na poskytování předpovědi pro jeden dokument za běhu programu či webové aplikace. Jak už jméno napovídá, tak je možné tímto přístupem získat predikce kdykoliv, což umožňuje okamžité obsloužení potřeb uživatelů. Příklady mohou zahrnovat našeptávání při hledání na webových stránkach nebo odhady doby doručení při objednání jídla pomocí UberEats (Patruno, 2019). Využití pro účely kategorizace zákaznických požadavků spočívá například v dynamickém doporučování kategorie uživateli při vytváření nového požadavku.

Při zvolení tohoto přístupu je často vyžadována nízká doba odezvy, a proto je nutné zvolit správné technologie, které zajistí dostatečnou rychlost, typicky v řádu stovek milisekund. V tomto ohledu jsou systémy využívající online inferenci mnohem komplexnější než várková inference, jelikož je nutné brát v potaz odezvu všech operací pro provedení predikce, jako například sběr potřebných dat, transformace dat, rychlost algoritmu strojového učení, transformace výstupů atd. Na obrázku 5 je diagram principu inference v reálném čase, což spočívá ve vytvoření služby s REST API přístupovým bodem, který vrací předpověď na odeslaný požadavek (Patruno, 2019).

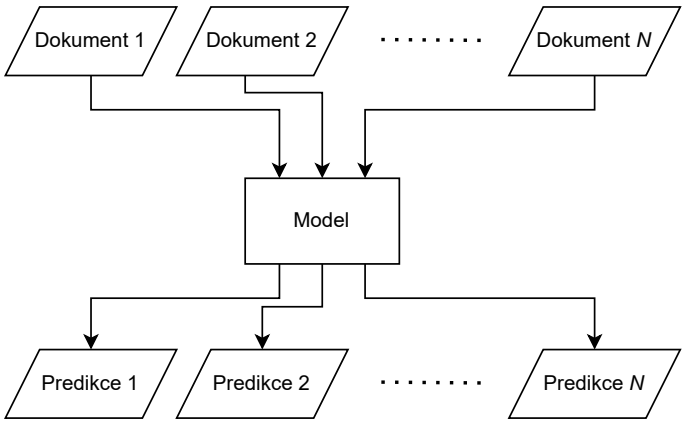


Obr. 5: Princip inference v reálném čase
Zdroj: Patruno, 2019

Offline inference

Offline inference se zabývá poskytováním předpovědí na várky neboli dávky dokumentů. Typicky se předpovědi generují na základě pravidelně se opakujícího rozvrhu, například po každé hodině nebo na konci každého dne. Získané předpovědi se poté uloží do databáze, z které se data mohou dále zpracovat a zpřístupnit uživatelům. Unikátní výhodou tohoto přístupu je využití technologií pro zpracování velkých dat pro urychlení generování predikcí. Příkladem může být doporučení produktů podle denní aktivity uživatele (Patruno, 2019). Využití pro účely kategorizace zákaznických požadavků spočívá například v detekci duplicitních tiketů na konci každého dne.

Ačkoli není architektura offline inference tak komplexní, jako online inference, existují dvě hlavní limitace, které mohou způsobit nežádoucí chování systému. První spočívá v relevanci předpovědí, kde je nutné nastavit rozvrh tak, aby předpovědi získané inferencí byly stále užitečné. Druhý je varianta tzv. problému studeného startu, což popisuje situaci, kdy predikce nejsou dostupné pro nová data, například při vytvoření nového účtu na eshopu s doporučením produktů na základě aktivity uživatele z předchozího dne (Microsoft, 2021). Na obrázku 6 je digram principu várkové inference, který spočívá v přímočarém načtení několika dokumentů, kde pro každý dokument model poté vrátí příslušnou předpověď.



Obr. 6: Princip várkové inference
Zdroj: Patruno, 2019

2.7.2 Optimalizace výkonu nasazení

Ať už se rozhodneme pro offline nebo online inferenci, výkon zvoleného modelu hraje hlavní roli v konečné použitelnosti řešení. Tento fakt se vztahuje zejména na aplikace online inference, kde je často nutná co nejnižší doba odezvy, nicméně do určité míry zasahuje i offline inferenci, kdy by nízký výkon modelu mohl vyprodukovat výsledky nad zastaralými a již nepodstatnými daty.

Existuje několik možností, jak výkon nasazeného modelu zvýšit. Mnozí poskytovatelé cloudových služeb nabízí automatické škálování nasazených služeb pomocí replikace služeb na několik virtuálních jednotek, což může zvýšit výkon nasazených služeb pro obsloužení více požadavků najednou (Kerner, 2021). Nicméně automatické škálování může pomoci pouze v situacích, kdy je model dostatečně rychlý pro obsloužení menšího množství požadavku. V situacích, kdy je inference za pomocí procesoru (CPU) nepřijatelně pomalá, je možné využít hardwarovou akceleraci pomocí grafického procesoru (GPU) pro zrychlení v oblasti 20násobku pro state-of-the-art modely založené na transformeru (Debut, 2019).

Další cesta zrychlení výkonu transformer odvozeného modelu je převod na otevřený formát modelů strojového učení ONNX (Open Neural Network eXchange),

který převádí strukturu transformeru na tzv. výpočetní graf. Po převodu je možné aplikovat různé optimalizace, specifické pro převáděnou architekturu (například BERT), které dále zvyšují výkon a snižují požadavky na operační paměť (Hugging Face, 2022b). Dále je možné využít tzv. kvantizace, která aproximuje hodnoty vah transformer modelu uložené ve formátu plovoucího bodu (FP32) do formátu celých čísel (INT8). Tato akce drasticky snižuje nároky na výkon a operační paměť, nicméně jelikož se jedná o aproximaci hodnot, mění se i původní přesnost modelu, což může v některých případech způsobit značnou ztrátu přesnosti (Li, 2020). V tabulce 3 je zobrazeno srovnání rychlosti inference PyTorch implementace BERT modelu oproti ONNX převedenému modelu s optimalizacemi, vše s využitím CPU inference. Lze zde vidět jasné zrychlení inference na všech délkách vstupu (v tabulce počet tokenů), ať už s využitím kvantizace nebo nikoliv. Dále lze vidět drastické snížení velikosti modelu až na čtvrtinu při využití ONNX kvantizace se zachováním a dokonce mírným zvýšením F1 skóre modelu.

Tab. 3: Srovnání modelu $\text{BERT}_{\text{BASE}}$ ve formátu PyTorch a ONNX při využití AVX2 (zkráceno)

	Rychlost inference [ms]			Velikost modelu [MB]	F1 skóre (512 tokenů)
	128 tokenů	256 tokenů	512 tokenů		
PyTorch (FP32)	69,14	124,03	283,11	418	90,18%
ONNX Runtime (FP32)	59,35	106,48	218,98	418	90,18%
PyTorch (INT8, kvantizace)	52,68	103,62	271,98	173	89,93%
ONNX Runtime (INT8, kvantizace)	46,54	96,18	214,7	106	90,20%

Zdroj: Li, 2020

2.7.3 Možnosti nasazení v rámci cloudových služeb

Jak už bylo zmíněno v úvodu práce, v dnešní době se mnoho firem připravuje na přechod do cloudového prostředí pro své výpočetní potřeby, což zahrnuje aplikace služeb pro zpracování přirozeného jazyka. Poskytovatelé cloudových služeb nabízí několik možností, jak model strojového učení nasadit. Pro účely této práce bylo využito cloudové platformy Microsoft Azure, a proto budou dále zmíněná řešení popisována v doméně Azure.

Azure disponuje speciálním prostředím pro usnadnění práce s modely strojového učení zvané Azure Machine Learning Studio. V rámci tohoto prostředí je možné, mimo jiné, snadno nasadit natrénované modely strojového učení, a to jak pro inferenci v reálném čase, tak pro várkovou inferenci. Co se týče možností pro inferenci v reálném čase, tak je zde nativně podporováno nasazení na Azure Kubernetes Service (AKS) nebo na Azure Container Instance (ACI). Rozdíl spočívá primárně v maximální propustnosti, podpory hardwarové akcelerace a možnostech škálování služby. Dokumentace doporučuje používat převážně AKS pro produkční nasazení, nicméně pro situace, kde není vyžadována vysoká propustnost, postačí ACI. Služby využívající AKS a ACI jsou účtovány po hodině bez ohledu na to, zda momentálně vykonávají kód nebo jsou nečinné, což může způsobit vyšší náklady za nasazenou službu (Microsoft, 2022a).

Využití Azure Machine Learning Studia není nutné pro vytvoření služby využívající strojové učení a Azure dále nabízí zajímavé možnosti pro nasazení modelů. Jednou z těchto možností jsou Azure Functions, které se snaží poskytnout jednoduché a nákladově přívětivé prostředí pro spouštění libovolného kódu. Na první pohled se zdají jako vhodné párování pro účely inference, jelikož je vesměs potřeba jedna hlavní funkce pro předpovězení výstupu na základě vstupů. Nicméně při využití tohoto přístupu je nutné počítat s několika limitacemi, které nejsou přítomny v řešeních pomocí AKS či ACI. Jednou z hlavních nevýhod je tzv. studený start funkce. Azure Functions fungují na serveless architektuře, která má možnost snížit své využívané prostředky na nulu, což drasticky snižuje náklady. Nicméně tato charakteristika zároveň způsobuje značné zvýšení doby odezvy, až o několik desítek sekund, při spouštění ze studeného stavu. Další nevýhodou je vynucené načtení všech inicializačních prostředků při každém spuštění funkce, což zahrnuje například načtení modelu strojového učení do paměti (Garg, 2020). Při využití velkých modelů, zejména modelů odvozených od transformeru, je nutno počítat s dalším zpomalením, jak je evidentní z tabulky 4.

Další možností nasazení je využití Platform-as-a-Service služby Azure App Service. Podobně jako Azure Functions, App Service poskytuje službu spouštění libovolného kódu, ale navíc podporuje využití Docker kontejnerů pro lepší přizpůsobení. App Service fungují jako mezikrok mezi Azure Functions a ACI/AKS, kde nemají nevýhody studeného startu a nucené inicializace prostředků při každém spuštění, jako Azure Functions, a zároveň je účtován pouze čas, kdy služba aktivně vykonává kód, což je dělá mnohem levnější na provoz než AKS/ACI. Nevýhodou oproti AKS/ACI je nižší propustnost (Microsoft, 2022b).

V tabulce 4 je zobrazené předběžné srovnání rychlosti služeb pro obsloužení jednoho požadavku při úkolu hledání nejpodobnějších tiketů ke vstupnímu tiketu.

Tab. 4: Srovnání rychlosti poskytovaných možností pro nasazení

	Doba obsloužení požadavku [ms]
Azure Functions (studený start)	32248
Azure Functions	19993
AKS	845
App Service	1433

3 Metodika

3.1 Použité technologie

Před stanovením a provedením různých experimentů je nutné vymezit nástroje, které budou tyto experimenty efektivně umožňovat. Prostředky pro vykonání experimentů se obecně dělí na hardware a software.

Co se týče hardwarových prostředků, tak byly v rámci této práce využity dvě platformy, které poskytují vzdálené prostředí pro vykonání kódu. Zejména bylo využito Azure Machine Learning Studia a bezplatné Google Colab platformy. V Azure Machine Learning Studiu bylo využito výpočetních instancí Standard D11 V2 pro experimenty nevyžadující hardwarovou akceleraci pomocí GPU, a Standard NC6 pro situace, kde bylo využití GPU nutné. Prostředí Google Colab bylo využito zejména v případech, kdy bylo nutné využít výkonnější GPU akceleraci (NVIDIA Tesla T4, 16 GB VRAM) než ta, která je poskytována v rámci předplatného v Azure Machine Learning Studiu (NVIDIA Tesla K80, 12 GB VRAM). Důvodem pro toto rozložení je minimalizace nákladů ze stránky Azure, jelikož se účtuje každá hodina, kdy jsou výpočetní instance zapnuté, a zároveň zamezení přesahování bezplatných limitů na Google Colab. Předplatné pro Microsoft Azure bylo poskytnuto v rámci spolupráce s firmou ALVAO s.r.o.

Ze softwarové stránky bylo využito skriptovacího jazyka Python 3 pro realizaci všech experimentů a vytvoření služeb pro nasazení na cloudovou platformu. Hlavní důvod pro využití Python 3 je široká podpora procesů strojového učení a snadné vytvoření služeb pro eventuální nasazení do Azure. Dále bylo využito několika knihoven, které implementují různé algoritmy strojového učení a zjednodušují jejich uživatelské rozhraní.

Pro získání dat z interní ALVAO databáze bylo využito knihovny *pyodbc* s ovladačem *Microsoft ODBC Driver 17 for SQL Server*. Jako základ pro procesy strojového učení zde sloužila knihovna *Scikit-learn*, která poskytuje různé nástroje pro transformaci textových dat, klasifikační algoritmy a vyhodnocení klasifikace. Pro snadnější manipulaci vstupních dat bylo využito knihovny *Pandas*. Pro experimenty, založené na modelu transformer, bylo využito zejména knihovny *Hugging Face Transformers* se základem *PyTorch*. Tato kombinace poskytuje určitou abstrakci pro práci s modely hlubokého učení a zejména modely odvozenými od transformeru. V některých experimentech bylo také využito knihovny *Simple Transformers*, která poskytuje další abstrakci nad knihovnou *Hugging Face Transformers* pro snadné a rychlé využití transformer modelů. Další výhoda této knihovny spočívá v intuitivní optimalizaci hyperparametrů pomocí spojení s online platformou Weights & Biases[2] při doladění transformer modelů. Pro experimenty využívající kosinovou podobnost pro zjištění podobných tiketů bylo využito knihovny *Sentence-Transformers*. Pro vytvoření služby pro nasazení na Azure bylo využito knihoven *Flask* a *Gunicorn*.

[2] https://wandb.ai/

3.2 Datové sady

Před stanovením samotných experimentů je nutné upřesnit, s jakými datovými sadami se bude pracovat. Použitá data mohou mít na výsledky experimentů významný dopad, kde některý přístup může poskytovat dobré výsledky s jednou datovou sadou, ale ne druhou. Proto budou v následujících podkapitolách popsány využité datové sady a jejich charakteristiky.

3.2.1 Interní dataset firmy ALVAO

Jako první využívaná datová sada je interní dataset firmy ALVAO. Jedná se o data z interního helpdesku obsahující zejména požadavky typu technické podpory v češtině. Jak už bylo zmíněno v úvodu, diplomová práce je úzce spojena s projektem modernizace helpdesku firmy ALVAO, a proto tato sada slouží jako hlavní datová sada pro účely diplomové práce, kde bude následně využívaná ve všech experimentech.

Požadavky jsou zde uloženy ve formě tiketů, které obsahují název požadavku a veškeré zprávy v rámci konverzace uživatele, systému a operátora. Zároveň je u každého tiketu stanovena jeho kategorie, kterou uživatel sám zvolí při vytváření požadavku. Dále je u většiny tiketů určen řešitel, kterého přiřadí operátor po zkontrolování požadavku.

Existuje zde tedy možnost využití celé konverzace, nicméně je nutné podotknout, že ne všechny zprávy budou nést důležité informace pro stanovení kategorie. Bude tedy nutné z tiketů vyjmout pouze významné zprávy a vypustit nepotřebné. Nicméně významné zprávy se liší podle daného experimentu. Jak bylo řečeno v kapitole 2.1, rychlé a správné stanovení kategorie tiketu je klíčové pro efektivní zákaznickou podporu, a proto například pro experiment stanovení kategorie tiketu bude významná pouze první zpráva žadatele pro co nejrychlejší kategorizaci. V jiném experimentu, například vyhledání a označení duplicitních tiketů, mohou být důležité všechny zprávy od žadatele, jelikož se v procesu konverzace může problém upřesnit, a vyžadovat jiné řešení, než by bylo zpočátku nutné. Samozřejmostí je odstranění všech systémových zpráv, jelikož nenesou užitečné informace pro stanovení kategorie, řešitele či duplicity k jinému tiketu.

Jednotlivé zprávy v tiketu mají formu e-mailu. Toto znamená, že odpovědi obsahují novou zprávu, a navíc původní zprávu, na kterou se odpověď vztahuje. Je tedy zřejmé, že bude nutné zprávy značně očistit a odstranit veškeré duplicitní zprávy uvnitř ostatních zpráv v rámci tiketu. V mnoha případech zprávy obsahují různé pomocné struktury, pozdravy, podpisy či závěry, které neobsahují žádné relevantní informace pro určení kategorie a pouze by znepřesnily vektorové reprezentace zpráv. Je tedy nutné tyto části zpráv odstranit. Dále je také nutné odstranit různé přílohy, jak už ve formě souboru, nebo ve formě textu (různé logy či chybové zprávy při pádu aplikace). Jedna zpráva tiketu a její očištění vypadá následně (anonymizováno):

Původní zpráva

```
Ahoj,
OA mi začal otvírat tikety v IE.

Najdi si tiket s navázanými tikety
Jdi na záložku Vazby a klikni na jeden z tiketů
Otevře se v IE, místo v OA

[image001.png]
Neděje se to vždy, ale je to čím dál častější

Jméno Příjmění | Pracovní pozice
```

Očištěná zpráva

```
OA mi začal otvírat tikety v IE.
Najdi si tiket s navázanými tikety
Jdi na záložku Vazby a klikni na jeden z tiketů
Otevře se v IE, místo v OA
Neděje se to vždy, ale je to čím dál častější
```

Jak už bylo řečeno, data obsahují také samotný název požadavku, který často poskytuje rychlé shrnutí problému či hrubou oblast, které se požadavek týká. Tyto informace mohou přinášet další porozumění požadavku jako takovému a je vhodné se zabývat i jimi. Názvy neobsahují žádné struktury, které by bylo nutno odstranit a lze s nimi pracovat přímo bez extensivního očištění. Název vypadá následně:

```
OA - otvírá tikety v IE, místo aby je rovnou zobrazil
```

Délka názvu požadavků odpovídá jejich charakteru. V tabulce 5 lze zpozorovat, že jedná se o převážně krátké texty v průměru dlouhé 51 znaků. Pro 99 % situací je délka názvu rovna nebo méně než 147 znaků, což odpovídá okolo 63 tokenům při využití Slavic BERT tokenizeru. V průměrném případě je počet tokenů mnohem menší a pohybuje se okolo 30 tokenů.

Co se týče délky jednotlivých zpráv, tak se většinou jedná o několik vět popisující problém uživatele. Nicméně existují zde delší texty obsahující několik odstavců. Tyto delší texty mohou způsobovat problémy pro vektorové reprezentace využívající transformer jako základ (například BERT), proto je nutné zjistit, jaké je rozložení délky textů zpráv. V tabulce 5 lze vidět, že 99 % prvních zpráv všech tiketů mají délku 1051 a méně znaků. Zároveň je vidět, že průměrná délka se pohybuje okolo 213 znaků. Pokud využijeme tokenizer modelu Slavic BERT, tak zjistíme, že zpráva o délce 1051 znaků odpovídá 488 tokenům, což stále spadá pod maximální počet tokenů (512), který Slavic BERT dokáže zpracovat. V situaci průměrné délky se zpráva vejde zhruba do 96 tokenů. Můžeme tedy konstatovat, že pro 99 % zpráv lze

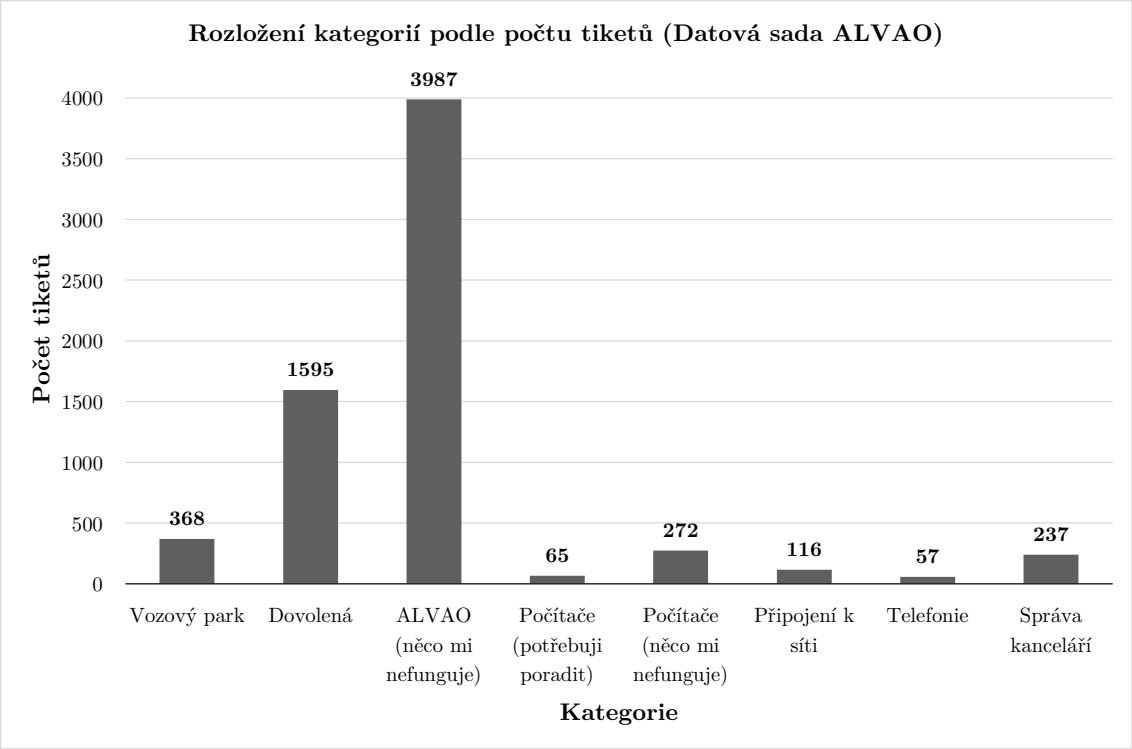
využít plný kontext pro vektorovou reprezentaci. Ve zbylém 1 % situací je nutno zprávu zkrátit na maximálně 512 tokenů, což způsobí menší ztrátu kontextu. Pokud bychom spojili název požadavku a první zprávu, dostaneme v průměrném případě 126 tokenů a pro 99 percentil 551 tokenů. V nejhorších případech je tedy nutné počítat s mírnou ztrátou kontextu.

Řešení problému ztráty kontextu není jednoznačné a záleží na dané situaci. V tomto případě se drtivá většina tiketů vejde do uváděného limitu, a proto zde není striktně nutné prozkoumávat metody přizpůsobení tiketů, aby se do limitu vešly. Také je poměrně nepravděpodobné, že by krátká přesahující část značně měnila kontext celého tiketu, alespoň pro účely kategorizace. U okrajových případů obsahujících opravdu velké množství znaků, a kde je přesahující část mnohem delší, záleží na charakteru daných okrajových případů. Často se v těchto případech jedná o různé struktury, které nebylo možné dostatečně očistit, jako například seznamy, textové přílohy, či různé logy. Pro tuto práci tedy budou zprávy pouze zkráceny. V případech, kde se zprávy do limitu nevejdou, lze využít více agresivního předzpracování textových dat. V některých případech je také možné rozdělit vstupní text na kratší celky, například věty, a ty vektorizovat samostatně a následně spojit. Tento způsob samozřejmě není možné využít při využívání klasifikační hlavy, a model by musel být doladěn na jiné úloze než klasifikace. Jak už bylo zmíněno, pro tento dataset není nutné tyto techniky využívat a nebudou dále zkoumány.

Tab. 5: Rozložení délky očištěných prvních zpráv v tiketu a názvů požadavků (Datová sada ALVAO)

	Počet znaků v první zprávě	**Počet znaků v názvu**
50. percentil	143	45
75. percentil	301	71
99. percentil	1051	147
Minimum	6	2
Maximum	10052	238
Průměr	213,7	51,4

Poskytnutý vzorek datové sady obsahuje celkem okolo 6700 unikátních tiketů, které jsou rozložené do 8 hlavních kategorií. Dále obsahuje 102 označených duplicitních párů a 52 odlišných řešitelů. V následujícím obrázku 7 lze vidět vizualizaci rozložení kategorií podle počtu tiketů. Z grafu je zřejmé, že se jedná o velice nevyvážený dataset, kde kategorie *ALVAO (něco mi nefunguje)* obsahuje téměř 60 % všech tiketů v datasetu. Zároveň kategorie *Počítače (potřebuji poradit)*, *Připojení k síti* a *Telefonie* dohromady obsahují méně než 4 % všech tiketů. Je zde vidět podobný trend jako u zkoumaných prací v kapitolách 2.6.2 a 2.6.3, kde se autoři taktéž setkávali s velice nevyváženými daty.



Obr. 7: Rozložení kategorií podle počtu tiketů (Datová sada ALVAO)

Kromě samotných kategorií tiketů jsou zde významní i jejich řešitelé, jelikož byl od firmy ALVAO vznesen zájem o doporučení řešitele na základě textu tiketu. Dataset má celkem 52 řešitelů, nicméně velké množství z nich řešilo malé množství tiketů. Konkrétně 32 řešitelů řešilo méně než 10 tiketů a 18 z nich řešilo právě jeden tiket. V tabulce 6 lze vidět zkrácený vývoj řešitelů v čase, kde je evidentní, že čas hraje klíčovou roli v doporučení řešitele. Tento jev dává smysl, jelikož řešitelé mohou odcházet z firmy nebo se mohou přesunout na jinou pracovní pozici. Nicméně zároveň tento fakt tvoří komplikace pro doporučení na základě samotného textu. Může se vyskytnout situace, kde se objevil nový problém, velice podobný starému, nicméně řešitel, který starý problém řešil, už ve firmě nepracuje a doporučit ho by nedávalo smysl. Z těchto poznatků je jasné, že samotný text tiketu nebude stačit pro kvalitní doporučení řešitele.

Tab. 6: Zkrácený vývoj řešitelů v čase (Datová sada ALVAO)

Rok \ ID řešitele	10665	718	680	11167	10409	11197	11310	1	12084	11961
2013	1	5	0	0	0	0	0	0	0	0
2014	0	25	148	2	4	0	0	0	0	0
2015	0	28	226	14	198	1	0	0	0	0
2016	9	36	26	31	513	0	1	0	0	0
2017	46	18	13	49	565	38	3	0	0	0
2018	36	0	7	198	228	46	3	175	1	0
2019	29	0	22	477	46	36	33	34	0	40
2020	39	0	78	19	28	15	520	0	0	48
2021	29	0	68	0	3	4	379	0	141	54
2022	7	0	19	0	2	1	41	0	44	16

3.2.2 Veřejný dataset technické podpory firmy Endava

Druhá využívaná datová sada je veřejný dataset firmy Endava. Jedná se o dataset technické podpory, který byl původně využitý firmou Microsoft pro účely tvorby webové služby pro automatickou kategorizaci tiketů v rámci Microsoft Azure (Żak et al., 2021). Dataset je v angličtině, což umožňuje využívat širší množství předtrénovaných modelů.

Podobně jako u interního datasetu firmy ALVAO jsou zde požadavky uloženy ve formě tiketů, které obsahují název a zprávy požadavku. Nicméně, kde se tento dataset liší, je v dostupnosti pouze první zprávy požadavku. Dataset byl už zprostředkován značné očištěný od irelevantních systémových zpráv, takže zde není nutné provádět dodatečné čištění. Podobně tak byly očištěny a anonymizovány samotné zprávy tiketů, kde autoři odstraňovali různé pomocné struktury, emailové hlavičky, stop slova, nealfanumerické znaky, konkrétní jména a další nežádoucí slova. Z krátkého výčtu čisticích operací můžeme vidět, že se jedná o poměrně extensivní zásah do textové struktury, a je možné, že tyto operace způsobily určitou ztrátu kontextu některých zpráv, což může snížit kvalitu embeddingů modelů založených na transformeru. Název a zpráva jednoho tiketu vypadá následně:

```
please action missing time

access card working hi access card stopped working can you please
help with thank you master en
```

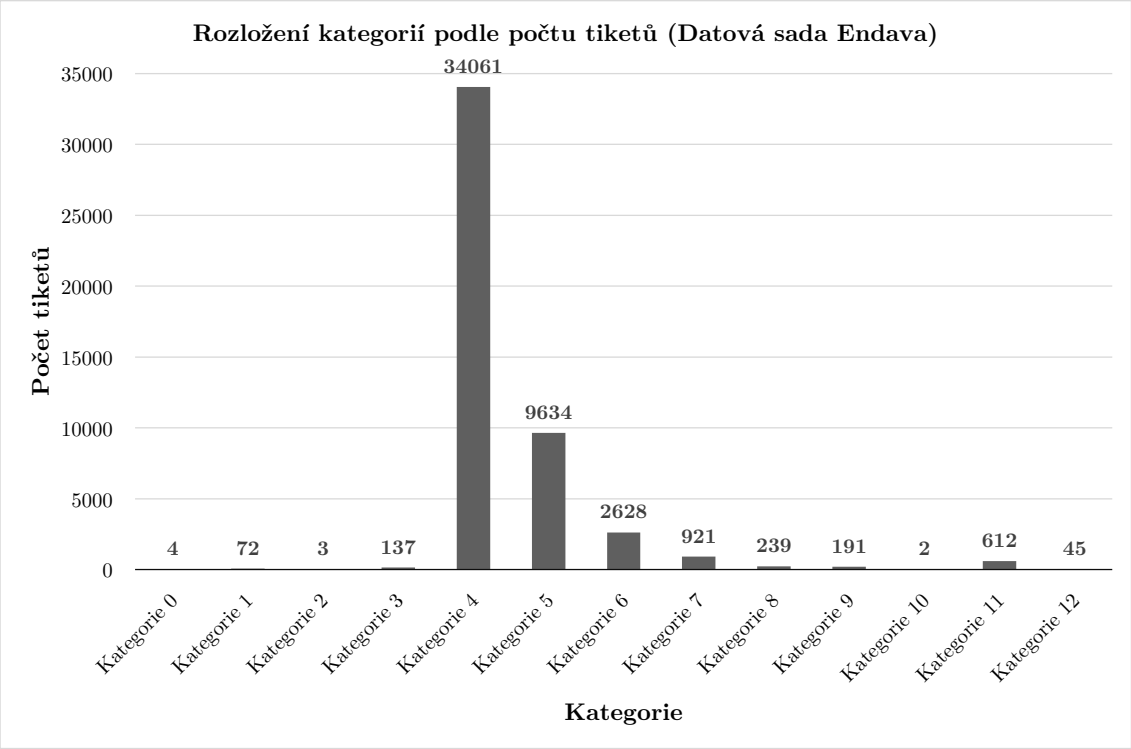
V tabulce 7 lze vidět rozložení délky zpráv a názvů. Je zde vidět, že oproti prvnímu zkoumanému datasetu jsou zprávy v průměru o zhruba 50 znaků delší a názvy v průměru o 30 znaků kratší. Také je zde velký rozdíl v 99. percentilu délek zpráv, kde 99 % všech zpráv má délku 1900 a méně znaků, což je značný rozdíl oproti ALVAO datasetu. Nicméně pokud využijeme tokenizer modelu $\text{BERT}_{\text{BASE}}$ tak

zjistíme, že se tikety o délce 1900 znaků vejdou do 436 tokenů, což je stále v rámci limitu 512 tokenů. I v případě, že se vyskytne tiket s délkou zprávy i názvu na horní hranici 99. percentilu, se vstup vejde do 455 tokenů a zajišťuje plný kontext. Vysvětlení tohoto jevu spočívá v jazyku textu a dodatečného očištění. U prvního datasetu se pracuje s češtinou, která je obecně komplexnější než angličtina a obsahuje více unikátních znaků. Proto nemůže tokenizer modelu BERT obsahovat delší tokeny a pro jednotlivá slova je často nutno použít více tokenů. V případě využití anglického datasetu Endava se naopak velké množství slov namapuje na právě jeden token, drasticky snižující celkový počet potřebných tokenů. Samozřejmě se i zde objevují odlehlé tikety s vyšším počtem znaků a pro tyto případy je nutné počítat s určitou ztrátou kontextu.

Tab. 7: Rozložení délky zpráv tiketu a názvů požadavků (Datová sada Endava)

	Počet znaků ve zprávě	**Počet znaků v názvu**
50. percentil	150	20
75. percentil	275	30
99. percentil	1900	68
Minimum	2	0
Maximum	7011	151
Průměr	266,6	22,9

Datová sada obsahuje celkem 48549 tiketů a 13 kategorií. Jména kategorií jsou navíc anonymizovaná. V grafu na obrázku 8 lze opět vidět, že se jedná o velice nevyvážený dataset, dokonce mnohem více než interní dataset firmy ALVAO. Dominantní kategorie 4 obsahuje přes 70 % všech tiketů a druhá nejvíce zastoupená kategorie 5 obsahuje téměř 20 % všech tiketů. Zároveň zde existují kategorie s opravdu nízkým počtem tiketů, jako například kategorie číslo 0, 2 a 10, které dohromady obsahují méně než 10 tiketů. Je tedy zřejmé, že zde bude nutno využít některé techniky vyvážení dat pro zkvalitnění výsledných klasifikátorů.



Obr. 8: Rozložení kategorií podle počtu tiketů (Datová sada Endava)

3.3 Experimenty

3.3.1 Přiřazení kategorie k tiketu

Přiřazení kategorie k tiketu slouží jako hlavní experiment této práce, a proto bude tvořit největší část všech experimentů. Jedná se o přiřazení neboli predikci kategorie tiketu na základě jeho textu. Vstupem do tohoto experimentu budou všechny zmíněné datasety v kapitole 3.2, přičemž pro každou datovou sadu budou všechny postupy provedeny odděleně za účelem získání hlubšího porozumění různých postupů nad různými datovými sadami ze stejné oblasti.

První zkoumaná datová sada bude již zmíněný interní dataset firmy ALVAO. Textová data byla očištěna způsobem popisovaným v kapitole 3.2.1 a následně byl spojen název požadavku a jeho první zpráva. Důvod pro spojení názvu a první zprávy spočívá v zapojení co nejvíce vstupního textu od uživatele při vytváření požadavku. Navíc se z předběžných experimentů ukázalo, že název požadavku poskytuje velké množství informací pro kategorizaci. Spojení bylo provedeno konkatenací názvu požadavku a první zprávy, přičemž na konci názvu byla přidána tečka kvůli lepšímu zařazení pro transformer odvozené modely. Příklad připraveného dokumentu vypadá tedy následně:

```
OA - otvírá tikety v IE, místo aby je rovnou zobrazil. OA mi začal
otvírat tikety v IE.
Najdi si tiket s navázanými tikety
Jdi na záložku Vazby a klikni na jeden z tiketů
Otevře se v IE, místo v OA
Neděje se to vždy, ale je to čím dál častější
```

Druhá zkoumaná sada bude dataset Endava. Jelikož se jedná o již důkladně očištěný dataset, tak žádné další čistící kroky nebudou prováděny. Pouze se spojí název požadavku a jeho první zpráva podobným způsobem, jako u ALVAO datasetu.

Reprezentace textových dat

Jak lze vidět z prací popisovaných v kapitole 2.6.2, tak jsou tradiční formy textové reprezentace stále relevantní. Proto budou jednou z hlavních zkoumaných forem reprezentace a zároveň budou sloužit jako základ, s kterým se budou modernější postupy srovnávat. Konkrétně se jedná o reprezentaci pomocí řídkého vektoru bag-of-words s TF-IDF vahami. Dále budou prozkoumány reprezentace využívající word embeddingy, konkrétně FastText a BERT. Cílem využití těchto modelů je zjištění přínosu kontextových embeddingů ve srovnáním se statickými.

Pro efektivní vektorizaci pomocí TF-IDF je nutno text dále připravit do formy, která umožňuje snadný převod do finální podoby, a zároveň neobsahuje zbytečná a nežádoucí slova či znaky. Jelikož jsou data z datové sady Endava důkladně předzpracována, tak se toto dodatečné zpracování bude týkat pouze datasetu ALVAO. Proces dodatečného předzpracování textu pro TF-IDF reprezentaci byl následný:

1. Záměna znaku nového řádku `\n` za mezeru
2. Odstranění všech nealfanumerických znaků
3. Převod na malá písmena
4. Rozdělení textu na jednotlivá slova
5. Odstranění stop slov
6. Zpětná konkatenace slov jako příprava pro TfidfVectorizer

Připravený tiket poté vypadá následně:

```
oa otvírá tikety ie místo aby rovnou zobrazil oa mi začal otvírat
tikety ie najdi si tiket navázanými tikety jdi záložku vazby klikni
tiketů otevře ie místo oa neděje čím častější
```

Použitá česká stop slova byla stažena pomocí knihovny *stop-words*. Po předpřipravením tiketů byl využit *TfidfVectorizer* z knihovny Scikit-learn s parametry `use_idf=True` a `lowercase=False`. Výsledkem je řídká vektorová reprezentace

o velikosti 24 674 dimenzí. Byly také prováděny krátké experimenty využívající stematizaci a nejčastější bigramy a trigramy, nicméně tyto metody poskytovaly mírně horší výsledky než pouze samotná slova.

FastText a ostatní modely produkující statické word embeddingy produkují reprezentace na základě natrénovaného jazykového modelu, který je nutno předem naučit. Pro získání kvalitních reprezentací slov je potřeba velké množství dat, a z tohoto důvodu bude pro trénování FastText jazykového modelu využito všech zpráv tiketů. Bylo zde zkoušeno i dotrénování volně dostupných předtrénovaných FastText modelů, nicméně experimenty s nově natrénovanými modely dosahovaly lepších výsledků. Pro účely této práce byla využita implementace z knihovny *fasttext*, kde byla použita funkce `fasttext.train_unsupervised` s parametry `epoch=25`, `dim=300` a `model="cbow"` pro natrénování jazykového modelu. Výsledný model poté poskytuje embeddingy pro jednotlivá slova dokumentu o velikosti 300 dimenzí. Jelikož je nutné tiket reprezentovat jedním všezahrnujícím vektorem, tak se jednotlivé word embeddingy následně zprůměrují, čímž získáme vektorizovaný tiket.

Poslední zkoumaná reprezentace textových dat využívá model BERT či jeho deriváty. Jelikož se jedná o model produkující kontextové word embeddingy, tak se nepoužívají další předzpracovací techniky, jako bylo potřeba pro předešlé reprezentace. Použití těchto technik by mohlo vést ke ztrátě kontextu v některých případech, a proto se zde využívá pouze základní očištění – popisováno v kapitole 3.2. Jazyk vstupních dat poté diktuje, jaký model typu BERT bude využit. Pro interní dataset firmy ALVAO bude využit model Slavic BERT, jelikož poskytuje mnohem lepší výsledky než původní vícejazyčná varianta modelu BERT (Arkhipov et al., 2019). Pro dataset Endava bude použit model $\text{BERT}_{\text{BASE}}$. Podobně jako u modelů produkující statické embeddingy, BERT pro generování embeddingů používá jazykový model, který je nutné předem natrénovat. Typicky se využívá již předtrénovaný model, který se případně dále doladí na konkrétním problému z důvodu vysokých nároků na výpočetní výkon. Jelikož jsou modely $\text{BERT}_{\text{BASE}}$ i Slavic BERT předtrénované, tak je možné je využít pro generování embeddingů tokenů bez jakéhokoliv dalšího doladění. Nicméně, podle Sun et al. (2019), dotrénování na koncovém úkolu bývá často prospěšné pro kvalitu výsledných embeddingů pro danou úlohu, a z tohoto důvodu budou zkoumány varianty bez doladění a s doladěním.

Samotný proces doladění modelu typu BERT obnáší několik otázek ohledně způsobu doladění a zvolených dat. Předtrénovaný Slavic BERT byl trénován na stejných úkolech jako originální BERT z původního návrhu, tedy na úlohách MLM a NSP. Je tedy zřejmé, že oba používané modely není možné využít pro kategorizaci tiketů bez přidání a dotrénování klasifikační hlavy. Nicméně je možné extrahovat embeddingy poslední vrstvy před hlavou řešící konkrétní úkol, které lze pak využít pro natrénování externího klasifikátoru nevyužívajícího hlavu BERTa. Zde se poté objevuje otázka, jak embeddingy tokenů zkombinovat pro vytvoření všezahrnujícího embeddingu tiketu. Podle Choi et al. (2021) existují vesměs dva hlavní způsoby, a to: průměr či sečtení všech embeddingů tokenů v rámci dokumentu, anebo využití

embeddingu speciálního [CLS] tokenu, který je optimalizován na základě dané předtrénovací či dolaďovací úlohy. Budou tedy testovány tři hlavní varianty vektorové reprezentace s využitím modelu BERT a to:

- předtrénované embeddingy z poslední vrstvy modelu (zprůměrované i pouze [CLS])
- doladěné embeddingy z poslední vrstvy modelu (zprůměrované i pouze [CLS])
- výstup doladěného modelu s klasifikační hlavou

Pro účely této práce bylo využito knihovny Simple Transformers pro přidání klasifikační hlavy a doladění modelu, přičemž byl dále kladen vyšší důraz na optimalizaci hyperparametrů.

Jak už bylo zmíněno v kapitole 2.5.1, BERT se skládá z několika za sebou poskládaných vrstev kodérů. V rámci těchto kodérů se ladí váhové matice, které poté do značné míry určují výslednou formu embeddingu tokenů. Při procesu doladění je možné některé vrstvy kodérů trénovat s odlišnou hodnotou α neboli rychlosti učení. Dále je možné vrstvy tzv. zamrazit, což znamená, že se některé vrstvy zcela vypustí z procesu doladění a nechají se jim původní předtrénované hodnoty. Zamrazené hodnoty mají poté hodnotu α rovno 0. Myšlenka na využívání odlišných rychlostí učení pro různé vrstvy transformeru spočívá ve využití kvalitní předtrénované základny pro nižší vrstvy a doladění vyšších vrstev na danou jazykovou doménu využívaných dat, což může přinášet kvalitnější výsledky. Další výhoda zmrazení vrstev spočívá v nižších nárocích na výpočetní výkon vynaložený pro doladění modelu. Dá se tedy využít hodnot α každé vrstvy BERTa jako hyperparametr, který je následně možné optimalizovat (Lee et al., 2019). Jednotlivé hodnoty α vrstev budou laděny po dvojicích, jelikož se neočekává příliš odlišná optimální hodnota mezi sousedními vrstvami. Další významné hyperparametry pro doladění modelu BERT jsou rychlosti učení přidaných hlav a počet epoch.

Pro samotné uskutečnění hyperparametrové optimalizace bude využita knihovna Simple Transformers a její napojení na platformu Weights & Biases. Parametry budou optimalizovány pomocí Bayesian prohledávání, kde pro každý pár vrstev budou zkoušeny hodnoty α mezi 0 a $1,8\text{e}{-4}$. Pro vrstvy klasifikační hlavy budou prohledávány hodnoty α mezi 0 a $1\text{e}{-3}$.

Vyvážení datasetů

Jelikož se pracuje s velmi nevyváženými datasety, tak zde budou zkoumány různé techniky vyvážení. Podle Aggarwal (2020) existují dvě hlavní metody pro vyvážení datové sady, a to podvzorkování dominantních kategorií a převzorkování menšinových kategorií. Aggarwal dále zvýrazňuje metodu SMOTE pro převzorkování kategorií, která vytváří syntetické vzorky po vektorizaci dokumentů. Dále se podle Coulombe (2018) v posledních letech objevuje další technika pro zvýšení reprezentace menšinových tříd, která se zabývá augmentací datové sady pomocí zpětného

překladu. Proces augmentace spočívá v překladu textu do jiného jazyka a následného překladu do původního jazyka vstupního textu. Myšlenka za tímto procesem spočívá v tom, že se v rámci strojového překladu nahradí některá slova za jejich synonyma či podobné výrazy. Výsledkem je tedy nový „syntetický“ tiket, který je velice podobný původnímu.

V této práci tedy budou zkoumány dopady odstranění menšinových kategorií, náhodné podvzorkování dominantních kategorií, náhodné převzorkování menšinových kategorií a augmentace menšinových kategorií pomocí strojového překladu. Pro převzorkování bude zkoumán efekt duplikace existujících tiketů a metody SMOTE pro vytvoření syntetických vzorků po vektorizaci. Pro dataset ALVAO nebude zkoumáno odstranění menšinových kategorií, jelikož neobsahuje kategorie s kriticky nízkým počtem tiketů. Pro dataset Endava bude zkoumán efekt odstranění kategorií číslo 0, 2 a 10, které dohromady obsahují pouze 9 tiketů.

Pro strojový překlad bude využito překladače Microsoft Translator uvnitř Azure, který poskytuje REST API rozhraní pro připojení a překlad z prostředí Pythonu. Pro augmentaci datasetu ALVAO budou augmentovány kategorie obsahující 150 a méně tiketů pomocí jazyků angličtina, francouzština a němčina. Pro augmentaci datasetu Endava budou augmentovány kategorie obsahující 250 a méně tiketů pomocí jazyků francouzština, němčina a španělština.

Pro obě metody převzorkování budou převzorkovány stejné kategorie, jako u augmentace. Pro dataset ALVAO budou tedy převzorkovány kategorie se 150 a méně tikety na 250 tiketů. Pro dataset Endava budou převzorkovány kategorie se 250 a méně tikety na 400 tiketů. Co se týče podvzorkování, tak budou pro ALVAO dataset podvzorkovány kategorie ALVAO (něco mi nefunguje) z 3987 tiketů na 1500 a Dovolená z 1595 tiketů na 1000. Pro datovou sadu Endava budou podvzorkovány kategorie číslo 4 a 5 na 2628 tiketů, tedy na stejný počet tiketů, jako kategorie číslo 6.

Vyvažovací metody se dále budou skládat, pokud experimenty s nimi dosahují lepších výsledků než s původním datasetem. Výjimkou je skládání obou metod převzorkování a augmentace, jelikož se jedná o metody zaměřující se na stejný problém – zvýšení reprezentace menšinových kategorií.

Zkoumané klasifikátory

Před stanovením jednotlivých klasifikátoru je nutné specifikovat rozdelění datových sad pro trénování, testování a popřípadě validaci. Pro účely této práce bylo využito trénovací/testovací rozdělení v poměru 85 % : 15 %. Výjimkou tohoto pravidla bude případ využití celého datasetu Endava, kde budou nejdříve podvzorkovány dominantní kategorie v testovací části, což změní poměr na zhruba 96 % : 4 %. V ostatních případech využití datové sady Endava bude rozdělení opět 85 % : 15 %. V předběžných experimentech se projevilo výrazné zlepšení výsledků při využívání podvzorkování na trénovací i testovací část, a proto bude podvzorkována testovací část i při využití celého datasetu, aby bylo možné přímo srovnávat výsledky

jednotlivých experimentů na stejné testovací části. V případech optimalizace hyperparametrů bylo využito k-křížové validace nad trénovacími daty s k rovno 4. Dále bylo nastaveno manuální semínko rovno 42 pro reprodukovatelnost experimentů.

Pro speciální případ, kde byl doladěn model BERT a jeho klasifikační vrstva, se využilo trénovacího/testovacího rozdělení uvedeného výše. V případech optimalizace hyperparametrů při doladění modelu BERT se dodatečně využilo trénovací/validační/testovací rozdělení v poměru 75 % : 10 % : 15 %. Důvod využití validačního datasetu oproti k-křížové validaci pro tento případ spočívá ve vysokých výpočetních nárocích, které značně prodlužují trénování k modelů pro k-křížovou validaci.

V rámci tohoto experimentu bude zkoumáno několik klasifikátorů, které se osvědčily jako vhodné pro kategorizaci textových dat do několika kategorií. Několik z uvedených klasifikátorů bylo využíváno v předchozích pracích, které byly zkoumány v kapitolách 2.6.2 a 2.6.3. Pro všechny klasifikátory, s výjimkou XGBoost, byla využita jejich implementace v rámci knihovny Scikit-learn. Pro XGBoost byla využita samostatná knihovna *XGBoost*. Pro klasifikaci využívající klasifikační vrstvu modelu BERT byla využita knihovna Simple Transformers. Pokud nebylo řečeno jinak, tak byly klasifikátory vždy natrénovány s výchozími parametry dané implementace. Konečný seznam všech zkoumaných klasifikátorů vypadá následně:

- XGBoost – Gradient Boosted Tree
- Support Vector Machine (SVM)
- Decision Tree
- Multinominal Naïve Bayes (MNB)
- Logistic Regression
- Random Forest
- K–Nearest Neighbours (KNN) – K = 3, 4, 5
- Multi Layer Perceptron (MLP) – 3 skryté vrstvy, počet neuronů ve skrytých vrstvách: 100, 200, 100
- Klasifikační hlava modelu BERT

Uvedené klasifikátory budou zkoumány se všemi využívanými textovými reprezentacemi, tedy TF-IDF, FastText a BERT vektory. U klasifikátorů poskytujících nejlepší výsledky budou dále optimalizovány jejich hyperparametry. Také bude zkoumán vliv kombinace nejvýkonnějších klasifikátorů do hard voting ensemble, kde bude výstupem kategorie s nejvíce hlasy.

Jako hlavní metrika pro ověření kvality klasifikátorů bude sloužit makro průměrované F1 skóre. Důvod pro využití makro průměru spočívá ve stejné důležitosti všech kategorií. Zároveň zde bude zkoumáno sdružení F1 skóre pomocí váženého

průměru, jelikož jsou vstupní datasety velice nevyvážené. Tato metrika by mohla lépe zachytit kvalitu klasifikátoru ve více obyvklých případech.

Vizualizace výsledků

Pro vizualizaci výsledků bude využito zejména matice záměn, a to ve dvou provedeních: matice s konkrétními počty instancí pro každou kategorii a normalizovaná matice obsahující hodnoty od 0 do 1 pro každou kategorii. Důvod pro využití dvou matic spočívá v lepší vizualizaci kategorií s odlišným počtem instancí, kde můžou absolutní hodnoty instancí poskytovat zavádějící výsledky. Jelikož bude zkoumáno velké množství modelů, tak budou výsledky vizualizovány pouze u modelů poskytujících nejlepší výsledky.

Paměťové a časové nároky

Jelikož je konečným cílem tohoto experimentu vytvořit aplikaci nejlepšího výsledku pro reálné použití, tak je nutné brát v potaz nejen kvalitu výsledků ale i rychlost jejich získání. Z tohoto důvodu bude dále zkoumána velikost konečného modelu na disku, čas jeho načtení, rychlost získání predikce pro jeden dokument a maximální propustnost modelu za jednu minutu.

3.3.2 Doporučení řešitele pro tiket

Druhý zkoumaný experiment se zabývá doporučením řešitele pro nový tiket. V rámci tohoto experimentu bude použit pouze interní dataset firmy ALVAO, kde je obsaženo celkem 52 odlišných řešitelů. Problém doporučení řešitele se dá stále považovat jako kategorizační, kde se k tiketu přiřazuje kategorie ve formě řešitele.

Jak už bylo zmíněno v kapitole 3.2.1, rozložení řešitelů je velice nevyvážené, a z tohoto důvodu bude nutné se dále zabývat určitým očištěním či vypuštěním některých řešitelů ze vstupní datové sady. Pointou tohoto experimentu je doporučit řešitele na základě již existujících podobných tiketů a jejich řešitelů, a proto je evidentní, že u řešitelů řešících opravdu malé množství tiketů, nastávají výrazné problémy. V situacích, kde měl řešitel přirazeno malé množství tiketů nebo dokonce pouze jeden tiket, algoritmy strojového učení selhávají, jelikož nemají dostatek dat pro naučení charakteristik daného řešitele. Je tedy vhodné prozkoumat další cesty, které lépe akomodují nízké počty existujících tiketů jednotlivých řešitelů. Další varianta je tyto problematické řešitele zcela vypustit, což by mohlo zpřesnit predikce u zbylých, častěji přiřazovaných, řešitelů.

Další otázka, která se zde nabízí, je, jaká data tiketu využít pro doporučení řešitele. Samozřejmostí je samotný text tiketu, který bude vypadat identicky jako u přechozího experimentu popisovaného v kapitole 3.3.1, nicméně nabízí se zde využít další informace, které v předchozím experimentu nebyly dostupné nebo jinak irelevantní. Konkrétně je zde vhodné prozkoumat efekt využití času vytvoření tiketu a samotné kategorie tiketu. Jak bylo evidentní z tabulky 6, řešitelé často řeší tikety pouze v daném časovém úseku, a je možné, že se objeví podobný tiket starému,

který měl na starosti řešitel, který už není dostupný. Podobně se dá očekávat, že ne každý řešitel řeší tikety ze všech kategorií. Tento jev lze vidět na následující tabulce 8, kde například řešitel s ID 1 se zabývá pouze tikety z kategorie Dovolená, nebo řešitel s ID 718 se zaměřuje spíše na tikety z kategorie Vozový park.

Tab. 8: Rozložení řešitelů podle kategorie řešených tiketů (vypuštěni řešitelé s 10 a méně tikety)

	Kategorie							
ID řešitele	**Vozový park**	**ALVAO (něco mi nefunguje)**	**Správa kanceláří**	**Dovolená**	**Počítače (něco mi nefunguje)**	**Telefonie**	**Připojení k síti**	**Počítače (potřebuji poradit)**
630	20	0	0	0	0	0	0	0
10665	13	13	167	0	2	1	0	0
718	105	0	6	0	1	0	0	0
10	15	0	0	254	1	0	0	1
680	10	350	3	164	31	14	24	11
11	17	4	22	540	13	3	0	0
181	0	0	0	380	0	0	0	0
419	2	45	0	0	89	26	25	8
11167	0	788	0	0	2	0	0	0
10409	3	1458	2	0	63	7	34	20
11197	115	0	25	1	0	0	0	0
676	5	1	0	41	0	0	0	1
11310	0	979	0	0	1	0	0	0
1	0	0	0	209	0	0	0	0
12084	0	185	0	0	1	0	0	0
11961	0	51	1	0	51	3	32	20
11394	0	10	1	0	1	0	1	1
11755	0	18	1	0	0	0	0	1
12262	50	0	5	0	0	0	0	0
11175	0	26	0	0	0	0	0	0

Z předběžných testů, jejichž výsledky jsou k dispozici v tabulce 9, lze usoudit, že zapojení těchto dodatečných informací má na kvalitu výsledného doporučení značný význam. Proces zapojení času vytvoření tiketu a jeho kategorie, využívaný v těchto experimentech, bude popsán v následující sekci. Pro tento dataset je dokonce doporučení pouze na základě textu prakticky nepoužitelné, jelikož experiment pouze s textem tiketu dosáhl velice nízkého váženého i makro průměrováno F1 skóre rovno 54 % a 45,46 %. Z tohoto důvodu budou v extensivnějších experimentech brány v potaz i údaje o času vytvoření tiketu a jeho kategorie.

Tab. 9: Předběžné testy zkoumající význam dodatečných údajů pro doporučení řešitele (TF-IDF, SVM)

	Vážený průměr			Makro průměr		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Pouze text	54,90 %	55,67 %	54,00 %	48,61 %	44,42 %	45,46 %
Text + kategorie	57,41 %	57,29 %	56,21 %	57,14 %	50,35 %	52,28 %
Text + čas vytvoření	77,42 %	78,14 %	77,43 %	67,42 %	65,62 %	65,84 %
Text + čas vytvoření a kategorie	80,48 %	81,07 %	80,47 %	71,47 %	69,42 %	69,84 %

Dále je také vhodné prozkoumat míru dominance řešitelů v rámci jednotlivých kategorií a v různých časových úsecích. Pokud lze problém efektivně vyřešit prostým doporučením dominantního řešitele na základě kategorie a časového úseku, tak není nutné budovat složité klasifikační systémy. Poznatky z této analýzy mohou sloužit pro stanovení základu, s kterým se klasifikační algoritmy využívané v tomto experimentu mohou srovnávat. Dominance řešitelů v rámci kategorií a jednotlivých let lze vidět v tabulce 10. Hodnoty uvnitř tabulky zobrazují míru dominance dané kategorie v daném roce jedním řešitelem, uvedenou v procentech. Pod touto hodnotou je dále identifikátor dominantního řešitele. Lze zde vidět, že v několika případech jsou některé kategorie či dokonce všechny kategorie dominovány jedním řešitelem s více než 80 % všech tiketů. Objevují se i případy, kdy je tato dominance úplná. Pokud by se vybudoval algoritmus, který vždy doporučí dominantního řešitele na základě kategorie a roku, tak by dosáhl accuracy rovno 65,46 %. Zároveň pokud by se používal pouze rok pro tento algoritmus, tak by dosáhl accuracy rovno 57,12 %. Samozřejmě takovéto algoritmy jsou v reálném provozu prakticky nepoužitelné, nicméně výše uvedené metriky mohou být užitečné pro srovnání s algoritmy strojového učení.

Tab. 10: Dominance řešitelů v jednotlivých kategorií podle roku (míry dominance dané kategorie v daném roce jedním řešitelem)

Rok	Všechny kategorie	Vozový park	Správa kanceláří	ALVAO (něco mi nefunguje)	Dovolená	Počítače (něco mi nefunguje)	Telefonie	Připojení k síti	Počítače (potřebuji poradit)
2011	100% (630)	100% (630)	X	X	X	X	X	X	X
2012	100% (630)	100% (630)	X	X	X	X	X	X	X
2013	72% (630)	75% (630)	100% (10665)	X	X	X	X	X	X
2014	60% (680)	83% (718)	X	94% (680)	49% (11)	60% (419)	100% (680)	X	X
2015	40% (680)	78% (718)	X	48% (680)	51% (11)	44% (680)	60% (680)	80% (680)	100% (680)
2016	67% (10409)	73% (718)	50% (10665)	93% (10409)	46% (181)	35% (680)	38% (419)	67% (680)	100% (680)
2017	60% (10409)	58% (11197)	71% (10665)	89% (10409)	45% (11)	49% (10409)	55% (419)	60% (10409)	67% (10409)
2018	28% (10409)	88% (11197)	67% (10665)	49% (10409)	67% (1)	56% (10409)	80% (419)	78% (10409)	67% (10409)
2019	47% (11167)	80% (11197)	69% (10665)	84% (11167)	39% (11)	43% (419)	80% (680)	38% (419)	57% (10409)
2020	50% (11310)	41% (12262)	77% (10665)	87% (11310)	27% (680)	35% (11961)	43% (10409)	50% (11961)	62% (11961)
2021	38% (11310)	74% (12262)	82% (10665)	69% (11310)	42% (11)	61% (11961)	57% (419)	59% (11961)	64% (11961)
2022	23% (12084)	78% (12262)	78% (10665)	49% (12084)	39% (11)	58% (11961)	X	60% (11961)	100% (11961)

Vyvážení datasetu

I v tomto experimentu se pracuje s nevyváženým datasetem a budou prozkoumány stejné techniky vyvážení jako v předchozím experimentu. Tedy odstranění menšinových řešitelů, náhodné podvzorkování dominantních řešitelů, náhodné převzorkování menšinových řešitelů a augmentace menšinových řešitelů pomocí strojového překladu.

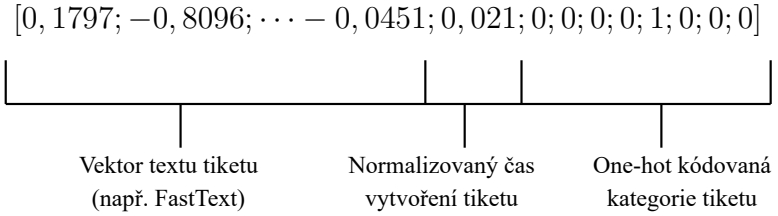
Ve všech zkoumaných případech budou odstraněni řešitelé řešící pouze jeden tiket, což snižuje počet řešitelů z 52 na 34. Dále bude zkoumán efekt odstranění řešitelů s 10 tikety a méně. Co se týče podvzorkování, tak bude zkoumán efekt podvzorkování dominantního řešitele s identifikátorem 10409 ze 1587 tiketů na 1000. Pro obě metody převzorkování budou převzorkování řešitelé se 150 a méně tikety na 250. Podobně tak bude provedena augmentace na řešitele se 150 tikety a méně pomocí stejných jazyků jako v předchozím experimentu, tedy angličtiny, němčiny a francouzštiny.

Reprezentace textových dat

Co se týče reprezentace textových dat, tak bude použito stejného přístupu jako u předchozího experimentu, zabývajícího se přiřazením kategorie k tiketu. Tedy budou zkoumány reprezentace pomocí bag-of-words s TF-IDF vahami, statické word embeddingy pomocí FastText a kontextové word embeddingy pomocí modelu Slavic BERT. Navíc zde bude prozkoumána reprezentace pomocí knihovny Sentence Transformers s předtrénovaným modelem *paraphrase-multilingual-MiniLM-L12-v2*, která umožnuje tikety převést do podoby, nad kterou lze provádět operace pomocí kosinové podobnosti, což by mohlo eliminovat problém nízkého počtu tiketů u některých řešitelů.

Tento experiment se bude mírně lišit ve způsobu využití již zmíněných netextových údajů. Pro většinu zkoumaných přístupů budou údaje přidány na konec vektoru tiketu. Čas vytvoření tiketu je v dodané datové sadě uložen ve formátu timestamp, který obsahuje datum a čas ve strukturované formě. Pro jednodušší manipulaci s tímto údajem bude nejdříve konvertován do tzv. POSIX formátu, což formát timestamp přetvaruje do jedné číselné hodnoty. Jedná se o formát zachycující počet sekund uplynutých od 00:00:00 1. ledna 1970. Jelikož se jedná o čísla v řádu stovek milionů, tak je dále nutné převedený čas vytvoření normalizovat, aby se zabránilo nerovnoměrném vlivu na některé klasifikátory, kde ostatní prvky vstupního vektoru mají hodnoty okolo nuly. Zároveň tato akce více zvýrazní rozdíly mezi časy vytvoření, jelikož jsou všechny hodnoty podobně daleko od absolutního začátku. Normalizace proběhla pomocí funkce `MinMaxScaler` z knihovny Scikit-learn, která číselné hodnoty přetvaruje do intervalu $\langle 0, 1\rangle$ s tím, že nejvyšší původní hodnotu zastoupí číslo 1 a nejnižší číslo 0.

Kategorie tiketu bude převedena na tzv. one-hot kódování a následně přiřazena na konec vektoru. Kódování one-hot spočívá ve vytvoření vektoru, který má stejný počet dimenzí jako počet kategorií s tím, že každá dimenze reprezentuje právě jednu unikátní kategorii. Vektor zakódované kategorie tedy obsahuje jednu jedničku na příslušné pozici a nuly na všech ostatních pozicích. Tato forma navíc zajišťuje, že se mezi kategoriemi nebudou tvořit nežádoucí vazby, kde by některé kategorie byly důležitější než jiné (Lewinson, 2020). Výsledný vektor vstupující do klasifikátoru vypadá následně:



U speciálního případu využití modelu BERT a jeho klasifikační vrstvy budou údaje převedeny do textové podoby a následně konkatenovány s původním textem tikety přes speciální token [SEP]. Podle Gu a Budkhar (2021) slouží tento přístup

jako velice dobrý základ pro kombinaci netextových a textových údajů při používání modelu BERT, zejména pokud je využito relativně malé množství netextových údajů. Jelikož se v tomto experimentu využívají pouze dva netextové údaje, tak bude využita pouze tato metoda. Vstup pro tento případ vypadá následně:

[CLS] Konkatenovaný název tiketu a očištěný text první zprávy [SEP]
2013-04-18 06:25:04.830 Kategorie: Telefonie [SEP]

V případě využití Sentence Transformers modelu pro přiřazení řešitele na základě podobných tiketů tyto formy kombinace dodatečných údajů nejsou vhodné a budou zakomponované při tvorbě algoritmu pro doporučení řešitele na základě podobných tiketů.

Zkoumané klasifikátory

Pro postupy využívající klasifikační algoritmy budou zkoumané stejné klasifikátory jako v předchozím experimentu z kapitoly 3.3.1. Pro speciální případ využívající Sentence Transformers model bude vytvořen algoritmus, který využívá embeddingy z předtrénovaného modelu zmíněné knihovny pro zjištění podobných tiketů. Zároveň bude v algoritmu figurovat čas vytvoření tiketu, nebo raději omezení časového úseku, který je pro zkoumaný tiket významný. Aby byl algoritmus dostatečně výkonný, tak je nutné mít všechny ostatní tikety převedené do vektorové podoby a uložené na dostupném místě. Jedná se o určitý index všech tiketů, který je pro účely této práce reprezentován datovou strukturou Python dictionary. Ta obsahuje klíče ve formě unikátních identifikátorů tiketů a hodnoty ve formě pole obsahujícího zejména vektorové reprezentace tiketů, čas vytvoření tiketů a jejich řešitele. Samotný proces doporučení řešitele spočívá v ohodnocení všech řešitelů, kteří řešili nalezené podobné tikety. Toto ohodnocení bude řešeno přiřazením tzv. důležitosti každého řešitele pro zkoumaný tiket, přičemž řešitel s nejvyšší důležitostí bude doporučen pro daný tiket. Důvod pro vypočítání důležitosti pro daný tiket oproti doporučení řešitele nejpodobnějšího tiketu spočívá ve snaze zvýšení priority menšinových řešitelů. Průběh algoritmu vypadá následně:

1. Vektorizuj vstupní tiket pomocí Sentence Transformers modelu
2. Omez index tiketů pouze na tikety starší 6 měsíců než vstupní tiket a novější 3 měsíce než vstupní tiket
3. Vyhledej 5 nejpodobnějších tiketů T v indexu tiketů pomocí kosinové podobnosti
4. Zjisti řešitele R nejpodobnějších tiketů
5. Pro každého řešitele R_i vypočítej jeho důležitost D_i
   $D_i = \frac{S_i}{P_i}$
   S_i – součet hodnot podobností řešitele R_i v množině T
   P_i – počet výskytů řešitele R_i v množině T

6. Vyber řešitele s nejvyšší hodnotou důležitosti

Časové omezení na tikety starší 6 měsíců bylo zvoleno na základě krátkých testů s různými hodnotami, kde hodnota 6 měsíců poskytovala nejlepší výsledky. V reálném využití se horní omezení novějších tiketů na 3 měsíce nevyužije, jelikož se počítá, že neexistují novější tikety než právě vytvořený tiket, nicméně bylo jej zde nutné přidat pro správný chod algoritmu při využívání historických dat, zejména prvních tiketů v datové sadě, pro které neexistovaly starší tikety. Výstupem algoritmu je jeden doporučený řešitel pro vstupní tiket. Lze zde tedy používat stejné metriky pro ohodnocení kvality predikce jako u ostatních klasifikačních algoritmů. Opět zde bude jako hlavní metrika zvolena makro průměrované F1 skóre a jako podpůrná metrika sdružení F1 skóre pomocí váženého průměru.

Vizualizace výsledků

Jelikož se stále jedná o kategorizační problém, tak budou výsledky vizualizovány stejným způsobem jako u předchozího experimentu zabývající se přiřazením kategorie k tiketu, tedy pomocí dvou matic záměn.

Paměťové a časové nároky

Výpočetní nároky budou zkoumány ze stejného hlediska jako u prvního experimentu, tedy velikost konečného modelu na disku, čas jeho načtení, rychlost získání predikce pro jeden dokument a maximální propustnost modelu za jednu minutu.

3.3.3 Vyhledání a označení duplicitních tiketů

Třetí a poslední zkoumaný experiment se zabývá vyhledáním a označením duplicitních tiketů. Tento experiment byl proveden na základě požadavku od firmy ALVAO, kde naráží na duplicitní tikety ve svém systému helpdesku. Duplicitní tiket zde znamená požadavek, který už je v systému helpdesku zaveden od jiného uživatele, či problém, který byl už řešen v rámci jiného požadavku. Samotný výskyt duplicitních tiketů lze snadno očekávat, jelikož se mohou vyskytnout situace, kdy nová chyba ovlivňuje několik uživatelů současně, a ti nezávisle na sobě vytvoří požadavek pro vyřešení té stejné chyby. Jelikož se jedná o zadání od firmy ALVAO, tak zde bude využit pouze jejich interní dataset, který obsahuje celkem 102 označených párů duplicitních tiketů.

V předchozích experimentech bylo využito pouze názvu požadavku a první zprávy od uživatele, jelikož se jednalo o experimenty zaměřené na kategorizaci při vytvoření tiketu, či těsně po vytvoření pro co nejrychlejší přiřazení kategorie a řešitele. Pro tento experiment je možné prozkoumat dva způsoby, a to: vyhledání duplicit při vytvoření tiketu, podobně jako u předchozích experimentů, a vyhledání duplicit po vytvoření tiketu, například na konci pracovního dne. Je zde vidět, že se jedná o dva různé způsoby vyřešení tohoto problému, které poskytují odlišné možnosti využití dat.

Pro první způsob je důležitá rychlost odhalení duplicit při vytvoření nového požadavku, a z toho důvodu bude možné srovnávat pouze první zprávy nově vytvořených tiketů s prvními zprávami ostatních tiketů, jelikož srovnání se všemi zprávami ve všech tiketech je velice výkonově náročné. Zároveň je pro tuto situaci dostupná pouze první zpráva, tudíž využívat další zprávy v rámci tiketu není možné. Pro druhý způsob jsou tyto nároky na výpočetní výkon mnohem méně restriktivní, jelikož se neočekávají výsledky co nejdříve. Jedná se o určitou formu dávkové inference. Je zde tedy možné srovnávat všechny zprávy nově vytvořeného tiketu se všemi zprávami ostatních tiketů, což navíc pomáhá řešit upřesňování požadavku, kdy se finální řešení může lišit od řešení z prvotní zprávy.

Reprezentace textových dat

Aby bylo možné najít podobné tikety či duplicity ke zkoumanému, tak je nutné text reprezentovat formou, která umožňuje aplikovat metriku podobnosti pro srovnání s jiným. Jedna forma, která tyto operace umožnuje, je tradiční bag-of-words s TF-IDF vahami, a proto zde bude zkoumán jako základ. Reprezentace pomocí TF-IDF bude získána stejnou metodou jako v prvním experimentu. Co se týče modelů založených na transformeru, tak se vyskytují problémy, které komplikují jejich efektivnost.

Původní model BERT nebyl navržen, aby produkoval embeddingy celých dokumentů, pro které lze snadně a efektivně využít tyto metriky podobnosti, a proto v tomto experimentu bude využita knihovna Sentence Transformers. Knihovna poskytuje předtrénované modely, které produkují embeddingy umožňující porovnání vektorizovaných dokumentů pomocí kosinové podobnosti. Konkrétně zde budou využity dva předtrénované modely, a to: *paraphrase-multilingual-mpnet-base-v2* a *paraphrase-multilingual-MiniLM-L12-v2*. Důvod pro využití dvou různých předtrénovaných modelů je převážně rychlost, kde autoři uvádí druhý předtrénovaný model jako 3krát rychlejší při ztrátě pouze necelých 2 % přesnosti na použité sadě testů (Reimers, 2022). Poskytované modely mají maximální limit tokenů 128, což drasticky snižuje délku vstupních zpráv, pro které je využito kontextu celé zprávy. Navíc tento fakt komplikuje využití všech zpráv v tiketu, kde bude nutné převést každou zprávu zvlášť a poté konkatenovat či průměrovat embeddingy všech zpráv v rámci tiketu.

Jak lze zjistit z názvu používané knihovny, tak se jedná o modely založené na transformeru, konkrétně na modelu BERT. Nabízí se zde tedy možnost doladění modelu pro kvalitnější stanovení podobnosti, nicméně existují určité požadavky, které znemožňují doladění předtrénovaných modelů nad zkoumanou datovou sadou. Aby bylo možné modely dotrénovat, tak je nutné mít k dispozici velké množství označených párů dokumentů a jejich procento podobnosti. Jelikož je v interním datasetu označeno pouze 102 duplicitních párů bez jakékoliv další míry podobnosti, tak je pro tento experiment efektivní doladění modelů bohužel nemožné.

Princip označení duplicitního tiketu

Jelikož se nejedná o striktně klasifikační problém, tak je nutné navrhnout způsob označení tiketu jako duplicitní k druhému. Tikety budou srovnávány pomocí kosinové podobnosti, která uvádí, jak jsou si dokumenty podobné. Podobnost je uváděna pomocí kosinu úhlu mezi srovnávanými vektory v intervalu $\langle -1, 1 \rangle$, přičemž vyšší hodnota znamená vyšší podobnost. Obecně je velice obtížné stanovit jednoznačnou mez, od které jsou tikety duplicitní, a proto zde budou vhodné tikety označovány jako kandidáti duplicity pro zkoumaný tiket, které mohou být dále filtrovány či ověřeny operátorem.

Pro tuto práci budou zkoumány dva způsoby. První způsob obnáší jednoduché zvolení tiketů s podobností 0,8 a více pro modely BERT a 0,2 pro TF-IDF, které se poté označí jako kandidáti duplicity pro zkoumaný tiket. Z předběžných experimentů bylo zjištěno, že tyto hranice poskytují dobré základy pro získání velice podobných tiketů. Důvod pro využití odlišných hranic pro BERT a TF-IDF spočívá v řídkosti vektorů produkovaných pomocí TF-IDF, kde kosinová podobnost poskytuje nižší hodnoty. Pro druhý způsob bude vytvořen algoritmus, který využívá relativní rozdíly podobnosti oproti statickému přístupu v prvním zkoumaném přístupu. Myšlenka za tímto přístupem je, že duplicitní páry si obecně budou více podobné než náhodné páry, nicméně míra podobnosti se bude lišit mezi jednotlivými duplicitními páry. Zároveň je možné, že nalezený nejpodobnější tiket ke zkoumanému nemusí být duplicitní, ten může mít mírně nižší hodnotu podobnosti než nejpodobnější, a proto je vhodné pracovat s tikety, které mají relativně blízkou hodnotu podobnosti k nejpodobnějšímu tiketu. Výsledný algoritmus vypadá následně:

1. Pro vstupní tiket T vypočítej kosinovou podobnost ke všem ostatním tiketům
2. Najdi tiket T_i s nejvyšší podobností S_{max} a přidej ho do pole kandidátů
3. Pokud tiket T_{i+1} má relativně nižší podobnost než T_i o maximálně 0,05, tak ho taktéž přidej do pole kandidátů a nahraď T_i za T_{i+1}
4. Bod 3 se opakuje, dokud podmínka platí a pokud zároveň pole kandidátů má délku méně než 4 (umožnění existence více duplicit)
5. Vypočítej průměrnou podobnost S_{avg} pro 300 náhodně vybraných textů
6. Pro každého kandidáta zjisti, zda má relativně vyšší podobnost než S_{avg} o 0,35 a výš. Pokud ne, tak odstraň tiket z pole kandidátů

Statické hodnoty podobnosti v algoritmu druhého způsobu byly zvoleny na základě krátkých experimentů s různými hodnotami. Metrika pro vyhodnocení kvality vypočítané podobnosti bude řešena pomocí vizualizace. Metrika pro jednotlivé experimenty vyhledání duplicitních kandidátů ke vstupnímu tiketu bude podíl všech označených duplicit interního datasetu ve vyhledaných kandidátech. Dále zde bude zkoumána metrika recall, která v tomto experimentu udává, zda byly ve vyhledaných kandidátech nalezeny označené duplicity.

Vizualizace

Vizualizace proběhne pomocí krabicových grafů, kde bude zkoumáno rozložení podobností duplicitních párů a náhodných párů. Čím oddálenější budou krabice duplicitních a náhodných párů a čím méně se budou krabice a jejich vousy protínat, tím kvalitnější bude hodnota podobnosti.

Paměťové a časové nároky

Jako v předchozích experimentech tak i zde budou zkoumány paměťové a časové nároky. Bude zde zkoumána velikost konečného modelu na disku včetně struktur pro uložení vektorů ostatních tiketů, čas jeho načtení, rychlosti získání duplicit pro jeden tiket a maximální propustnost modelu za jednu minutu.

3.4 Nasazení

Proces nasazení řešení ze zmíněných experimentů spočívá ve vytvoření služby integrující tato řešení a jejich modely pomocí knihovny Flask, kterou je poté možné nasadit na Azure. Výsledná služba poté poskytuje online inferenci neboli inferenci v reálném čase. Důvod pro zvolení tohoto typu inference jsou požadavky na predikci kategorie při vytváření nového tiketu. Zároveň tuto službu využívá chatbot v rámci stejného projektu, který potřebuje obstarat informace co nejdříve, zejména při vyhledávání duplicitních požadavků. Z předběžných experimentů bylo zjištěno, že nasazení na Azure přes PaaS platformu Azure App Service poskytuje nejlepší rovnováhu mezi náklady a rychlostí, a proto se tato kapitola bude zabývat procesem nasazení na tuto platformu.

Nicméně, než se vytvoří samotná služba, tak je nutné optimalizovat využívané modely typu BERT pro nasazení. Pro tyto účely bylo využito ekosystému ONNX, pro který má knihovna Hugging Face Transformers předpřipraven skript, který převede PyTorch model do formátu ONNX. Jedná se o skript `convert_graph_to_onnx.py`, který se spustí následujícím příkazem v příkazové řádce:

```
python convert_graph_to_onnx.py
        --framework pt
        --model <vstupní PyTorch model> <převedený ONNX model>
```

Je také možné provést kvantizaci modelu při převodu pomocí přepínače `--quantize`. Při využívání předpřipraveného skriptu pro převod nejsou aktivovány optimalizace, jelikož mnoho z nich záleží na konečném hardwaru, který bude kód vykonávat. Z tohoto důvodu je nutné buď optimalizace přímo přidat do modelu dodatečně před nasazením, nebo je aktivovat při inicializaci výsledné služby (Hugging Face, 2020).

Po připravení modelů je možné vytvořit samotnou službu. Jedná se o službu, která obsahuje REST API endpointy, na které je možné poslat požadavek pro predikci kategorie vstupního tiketu, doporučení řešitele a vyhledání duplicit. Služba

dodržuje obecnou strukturu aplikace v knihovně Flask, kde je rozhraní definováno v souboru `app.py`. Komunikace se službou je zprostředkována pomocí formátu pro výměnu dat JSON. Pro spuštění služby je využito knihovny Gunicorn. Hlavní soubor `app.py` má následnou strukturu:

```
1  from flask import Flask, request, jsonify
2  from nlp import predict_category, load_models, create_update_thread
3
4  def create_app():
5          app = Flask(__name__)
6          load_models()
7          create_update_thread()
8          return app
9
10 app = create_app()
11
12 @app.post("/categories")
13 def categories():
14         data = request.json
15         result = predict_category(data)
16         return jsonify(result), 201
```

Obr. 9: Struktura souboru app.py (zkráceno)

Aby bylo možné efektivně a rychle zjistit duplicitní požadavky k nově vytvořenému tiketu, tak je nutné mít všechny dosavadní tikety převedené do vektorové podoby a uložené na snadno a rychle dosažitelném místě. Pro tuto práci bylo zvoleno ukládání vektorů do operační paměti ve formě Python dictionary, ke kterým má služba velmi rychlý přístup. Tento postup je pro tuto práci paměťově proveditelný, jelikož se jedná o množství tiketů v řádu tisíců, nicméně v případech, kde by bylo množství existujících tiketů mnohem vyšší, by se musel zvolit jiný přístup. Pro průběžnou aktualizaci Python dictionary s uloženými vektory je ve Flask aplikaci při inicializaci vytvořeno další vlákno, které se periodicky dotazuje databáze a převádí nové tikety, které doposud nemá převedené a uložené. Proces aktualizace se provede i při inicializaci služby, kde se vektorizují všechny tikety v databázi.

Po vytvoření služby je využito platformy Docker pro převod Flask aplikace na soběstačný Docker kontejner. Tento krok zajišťuje snadné nasazení na App Service a zároveň umožňuje minimalizovat službu pouze na opravdu potřebné knihovny. Jako základní obraz byl využit *python:3.9.5-slim*, který je založen na operačním systému Debian 10. Důvod pro využití neaktuální verze Pythonu je kompatibilita s ovladačem Microsoft ODBC Driver 17 for SQL Server. Základní obrazy s novější verzí Pythonu jsou založeny na operačním systému Debian 11, pro který neexistuje kompatibilní verze ovladače. Jako cílová architektura při vytváření Docker obrazu byla zvolena linux/amd64.

Při převodu na Docker kontejner je nutno dávat pozornost samotnému modelu, konkrétně jeho umístění. Existují dvě hlavní možnosti, jak model přiložit ke službě, a to buď zahrnutí v samotném Docker obrazu, nebo připojení přes vzdálené úložiště při nasazení služby. První varianta má nevýhodu v tom smyslu, že pokud se využívá velký model, například BERT, tak výsledný Docker obraz bude zabírat velké množství místa, což způsobuje pomalejší spuštění služby. Zároveň tato varianta omezuje možnosti průběžného dotrénování modelu na nových datech, kde by bylo nutné vždy vytvořit nový Docker obraz s aktualizovaným modelem. Druhá varianta tyto problémy efektivně eliminuje, jelikož se připojení modelu provádí po spuštění Docker kontejneru. Azure poskytuje několik možností pro realizaci druhé varianty v rámci propojení Azure Storage účtu a ostatních prostředků jako například App Service. Pro tuto práci bylo zvoleno připojení modelu uloženého na Azure Files pomocí namountování vzdálené Azure File share.

Po připravení Docker obrazu je možné službu nasadit na Azure App Service. Princip spočívá v intuitivním postupu dle vytvářecího dialogu v portálu Azure. Důležité změny oproti výchozím nastavením vytvářecího dialogu zahrnovaly zejména zvolení nasazení pomocí Docker kontejneru a zvolení levnější cenové úrovně. Konkrétně byl zvolen plán B2, který má odhadované náklady okolo 21,55 EUR měsíčně. Je nutno podotknout, že pro možnost nasažení Docker kontejneru na Azure App Service je požadováno mít příslušný Docker obraz uložený na vzdáleném úložišti, jako například Docker Hub nebo Azure Container Registry. Pro účely této práce byla služba nasazena pouze v testovacím a debuggovacím rozsahu. Pro produkční nasazení by se kladl vyšší důraz na propustnost a škálovatelnost služby.

Po vytvoření samotné App Service je dále nutné nastavit několik parametrů. Nejdříve se musí zvýšit timeout limit pro spuštění Docker kontejneru. App Service má ve výchozích hodnotách nastavený timeout po 230 sekundách, nicméně jelikož inicializace služby, zejména načtení modelů a vektorizace existujících tiketů, může trvat déle než výchozí limit, tak je nutno jej navýšit na potřebnou hodnotu. Pro účely této práce byl limit navýšen na 1200 sekund. Následně je nutno nastavit namountování Azure File share s potřebnými modely strojového učení. Tato operace se provede následujícím příkazem v příkazové řádce Azure CLI:

```
az webapp config storage-account add
        --resource-group <jméno Azure resource groupy>
        --name <jméno App Service>
        --custom-id <jméno App Service>
        --storage-type AzureFiles
        --share-name <jméno File share>
        --account-name <jméno Azure Storage účtu>
        --access-key <klíč k Azure Storage účtu>
        --mount-path <cesta namountování v Docker kontejneru>
```

Po konečném nakonfigurování parametrů je služba nastavena pro prvotní inicializaci, po které je připravena obsluhovat požadavky pro kategorizaci nového tiketu, vyhledání duplicit a doporučení řešitele k tiketu.

4 Výsledky experimentů

Vzhledem k tomu, že se jedná o opravdu velké množství individuálních experimentů, celkem přes 1000 dílčích experimentů nepočítaje předběžné testování, tak budou v následujících kapitolách uvedeny pouze nejrelevantnější či nejlepší výsledky. Detailní výsledky všech dílčích experimentů jsou k dispozici v příloze A. Pokud nebylo řečeno jinak, tak byly všechny výsledky zkoumající výpočetní nároky získány pomocí CPU inference na platformě Apple M1 Pro s 16 GB RAM. V případě optimalizace hyperparametrů jsou jednotlivé hodnoty pro všechny relevantní modely dostupné v příloze B.

4.1 Přiřazení kategorie k tiketu

4.1.1 Datová sada ALVAO

V tabulce 11 lze vidět výsledky zkoumající efekt metod pro vyrovnání datové sady na datasetu ALVAO. Jedná se o nejlepší výsledky pro každou metodu vyrovnání. Obecně jsou všechny metody velmi blízko co se týče makro i váženého F1 skóre, nicméně augmentace datasetu pomocí strojového překladu dosahuje nejlepších výsledků. Zároveň je augmentace jediná metoda, která zlepšila makro F1 skóre oproti základu bez žádné metody.

Tab. 11: Srovnání metod vyrovnání datasetu (ALVAO)

			Makro průměr			Vážený průměr			
Metoda vyvážení	**Reprezentace**	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
Žádná	BERT (doladěn), průměr	XGBoost	84,43%	85,33%	84,51%	97,62%	97,41%	97,46%	97,41%
Augmentace	BERT (doladěn), CLS	XGBoost	87,96%	85,43%	86,38%	97,73%	97,71%	97,68%	97,71%
Převzorkování	BERT (doladěn), průměr	SVM	87,19%	84,26%	83,84%	97,70%	97,81%	97,58%	97,81%
Převzorkování (SMOTE)	FastText	SVM	90,22%	81,87%	83,38%	97,17%	97,01%	96,86%	97,01%
Podvzorkování	BERT (doladěn), průměr	KNN (3)	84,26%	84,31%	83,68%	97,30%	97,01%	97,08%	97,01%

Z vyšší tabulky lze vidět, že nejlepší výsledky u každé metody byly dosaženy s využitím dodatečně natrénovaného klasifikátoru. Tedy klasifikační hlava doladěného modelu BERT nikdy nedosahovala nejlepších výsledků, byť klasifikátory využívající jeho embeddingy ano. Výsledky klasifikační hlavy lze vidět v tabulce 12, kde bylo opět dosaženo nejlepších výsledků s augmentovaným datasetem.

Tab. 12: Srovnání metod vyrovnání datasetu – klasifikační hlava modelu BERT (ALVAO)

			Makro průměr			Vážený průměr			
Metoda vyvážení	**Reprezentace**	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
Žádná	BERT (doladěn)	Klasifikační hlava	77,85%	81,39%	79,15%	96,93%	97,31%	97,02%	97,31%
Augmentace	BERT (doladěn)	Klasifikační hlava	84,61%	85,03%	84,56%	97,32%	97,31%	97,29%	97,31%
Převzorkování	BERT (doladěn)	Klasifikační hlava	87,22%	83,38%	83,28%	97,76%	97,81%	97,59%	97,81%
Podvzorkování	BERT (doladěn)	Klasifikační hlava	73,77%	80,47%	76,50%	96,52%	96,52%	96,38%	96,52%

Je tedy zřejmé, že pro tento dataset je augmentace nejvhodnější metoda vyvážení a je dobré se na ni dále zamířit. V tabulce 13 lze vidět nejlepší výsledky jednotlivých textových reprezentací nad augmentovaným datasetem. Obě reprezentace využívající doladěný model BERT dosahují poměrně dobrých výsledků a zabírají 3 první místa. Samotná klasifikační hlava je mírně horší. Zajímavý poznatek je, že klasifikátory využívající embeddingy z nedoladěného modelu BERT poskytují mnohem horší výsledky než doladěný, a dokonce horší než TF-IDF v případě využívání pouze speciálního tokenu CLS.

Tab. 13: Nejlepší výsledky pro všechny vektorové reprezentace – augmentovaný dataset (ALVAO)

		Makro průměr			Vážený průměr			
Reprezentace	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
BERT (doladěn), CLS	XGBoost	87,96%	85,43%	86,38%	97,73%	97,71%	97,68%	97,71%
BERT (doladěn), průměr	XGBoost	85,63%	85,87%	85,47%	97,53%	97,51%	97,49%	97,51%
BERT (doladěn), CLS	KNN (3)	87,81%	85,43%	85,20%	97,70%	97,51%	97,46%	97,51%
BERT (doladěn)	Klasifikační hlava	84,61%	85,03%	84,56%	97,32%	97,31%	97,29%	97,31%
FastText	SVM	84,60%	81,95%	82,43%	96,86%	97,01%	96,85%	97,01%
BERT (nedoladěn), průměr	Logistic Regression	78,30%	77,65%	77,92%	95,84%	95,72%	95,77%	95,72%
TF-IDF	XGBoost	80,60%	73,76%	76,14%	95,23%	95,32%	95,17%	95,32%
BERT (nedoladěn), CLS	Logistic Regression	69,20%	68,88%	68,99%	93,87%	93,93%	93,90%	93,93%

Co se týče hyperparametrové optimalizace nejlepších klasifikátorů a jejich sloučení do voting ensemble, tak bylo dosaženo zanedbatelného zlepšení v obou případech. Pro optimalizaci zde byly vybrány klasifikátory využívající token CLS doladěného modelu BERT s algoritmy XGBoost, KNN a také jeho klasifikační hlava, jejíž

výsledky před optimalizací lze vidět v předchozí tabulce. Následné spojení těchto optimalizovaných klasifikátorů poskytovalo mírně lepší výsledky než nejlepší výsledek před optimalizací, nicméně samotný algoritmus XGBoost po optimalizaci poskytoval o necelé půl procento lepší makro F1 skóre než spojení. V tabulce 14 lze vidět detailní výsledky.

Tab. 14: Výsledky optimalizace hyperparametrů a sloučení do voting ensemble – augmentovaný dataset (ALVAO)

		Makro průměr			Vážený průměr			
Reprezentace	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
BERT (doladěn), CLS	XGBoost	90,23%	85,81%	86,94%	97,82%	97,61%	97,60%	97,61%
BERT (doladěn), CLS a klas. hlava	Voting Ensemble	89,33%	85,81%	86,55%	97,78%	97,61%	97,59%	97,61%
BERT (doladěn)	Klasifikační hlava	85,89%	85,59%	85,67%	97,51%	97,51%	97,51%	97,51%
BERT (doladěn), CLS	KNN (3)	87,81%	85,43%	85,20%	97,70%	97,51%	97,46%	97,51%

Výpočetní nároky jednotlivých přístupů lze vidět v tabulce 15. Je zde vidět, že přístupy poskytující lepší výsledky jsou obecně více výpočetně náročné, s výjimkou přístupu využívající FastText embeddingy a klasifikátor SVM, který dosahuje nejvyšší propustnosti. Je zde nutné podotknout, že modely BERT nejsou v této tabulce optimalizované. Efekt optimalizace bude prozkoumán v kapitole 4.4. Spojení voting ensemble očekávaně poskytuje nejnižší propustnost, nicméně velkou část zpomalení zde dělá klasifikační hlava modelu BERT, s predikcí pro jeden dokument trvající přes 2 a půl sekund.

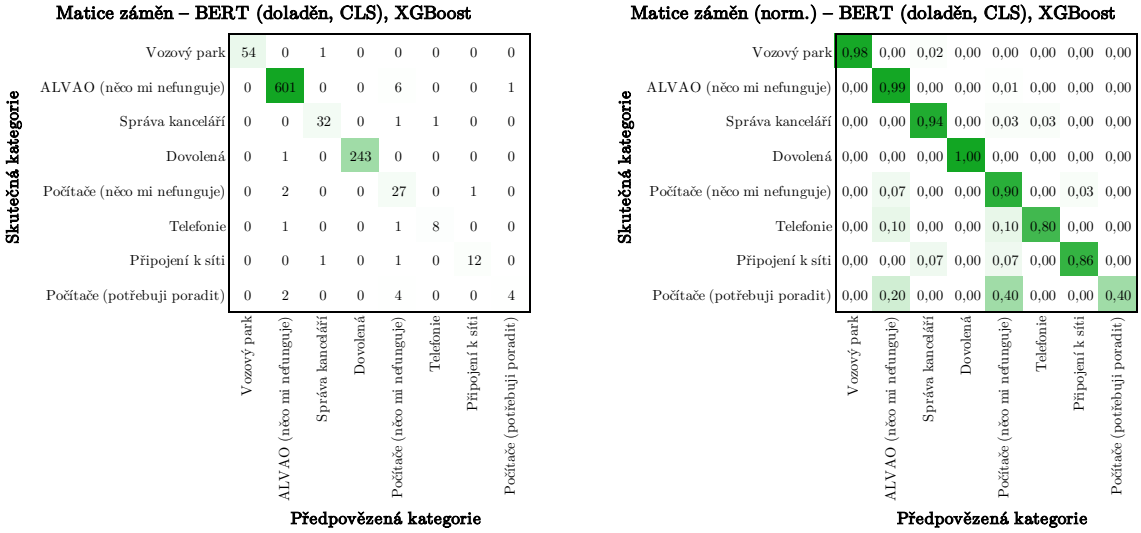
Tab. 15: Srovnání paměťových a časových nároků klasifikátorů (ALVAO)

Reprezentace	**Klasifikátor**	**F1 skóre (makro)**	**Velikost na disku [MB]**	**Čas načtení [ms]**	**Rychlost získání predikce pro 1 dokument [ms]**	**Maximální propustnost za 1 minutu [počet predikcí]**
BERT (doladěn), CLS	XGBoost	86,94%	716	1663	500	120
BERT (doladěn), CLS a klas. hlava	Voting Ensemble	86,55%	735,4	3769	3427	17,5
BERT (doladěn)	Klasifikační hlava	85,67%	715,8	1646	2630	22,81
BERT (doladěn), CLS	KNN (3)	85,20%	735,2	1683	643	93,31
FastText	SVM	82,43%	2491,92	1872	1,89	31746
TF-IDF	XGBoost	76,14%	0,665	1,11	2,24	26785

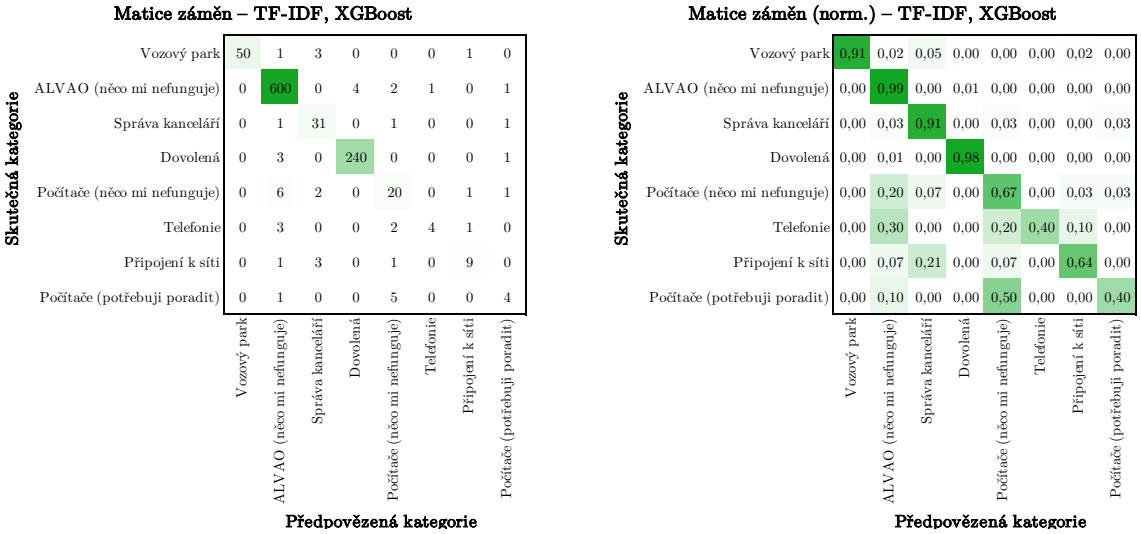
Matice záměn pro nejlepší výsledek lze vidět na obrázku 10. Obecně klasifikátor poskytuje velice dobré výsledky pro drtivou většinu kategorií, což je dále podpořeno váženým F1 skóre rovno 97,60 %. Kde má klasifikátor největší problém je u kategorie Počítače (potřebuji poradit). Jedná se o jednu z menšinových kategorií, která

je zároveň velice podobná kategorii Počítače (něco mi nefunguje) a lze vidět, že klasifikátor často nesprávně přiřazuje vzorkům právě tuto podobnou kategorii. V této situaci by mohlo být vhodné sloučit obě kategorie, což by zpřesnilo klasifikaci dále.

Pokud se poté podíváme na matice záměn základního přístupu využívající TF-IDF a klasifikátor XGBoost, vyobrazené na obrázku 11, tak lze vidět, že základní přístup má dále problémy s kategoriemi Telefonie, Počítače (něco mi nefunguje) a Připojení k síti. Je tedy zřejmé, že se vyplatí v této situaci využívat modernější přístupy založené na transformeru.



Obr. 10: Matice záměn klasifikátoru s embeddingy BERT (doladěn, CLS) a algoritmem XGBoost



Obr. 11: Matice záměn klasifikátoru s embeddingy TF-IDF a algoritmem XGBoost

4.1.2 Datová sada Endava

V tabulce 16 je zkoumán efekt metod vyvážení datasetu na datová sadě Endava. Opět jsou zde zobrazeny nejlepší výsledky pro danou metodu. Jak bylo řečeno v kapitole 3.3.1, tak zde byly jednotlivé techniky skládány na sebe, pokud poskytovaly lepší výsledky. Konkrétně zde byly skládány metody podvzorkování, odstranění menšinových kategorií a jedna z metod pro zvýšení reprezentace menšinových kategorií. Samotné podvzorkování a odstranění menšinových kategorií poskytovalo největší zlepšení – necelých 19 % makro váženého F1 skóre. Augmentace dále poskytovala nejlepší výsledky, co se týče zvýšení reprezentace menšinových kategorií, a zvýšila makro F1 skóre o další 4,2 %. Výsledky jsou opět dominovány reprezentací pomocí doladěného modelu BERT.

Tab. 16: Srovnání metod vyrovnání datasetu (Endava)

			Makro průměr			Vážený průměr			
Metoda vyvážení	**Reprezentace**	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
Žádná	BERT (doladěn)	MNB	53,99%	44,47%	47,26%	77,19%	75,11%	75,13%	75,11%
Podvzorkování	BERT (doladěn)	XGBoost	66,40%	48,14%	52,78%	80,52%	79,96%	79,42%	79,96%
Odstranění min. kat.	BERT (doladěn), CLS	MLP	68,75%	64,32%	65,99%	77,04%	77,19%	76,96%	77,19%
Augmentace	BERT (doladěn), CLS	XGBoost	79,22%	65,69%	70,19%	79,17%	79,23%	78,98%	79,23%
Převzorkování	BERT (doladěn), CLS	XGBoost	78,62%	62,36%	67,59%	77,53%	77,57%	77,23%	77,57%
Převzorkování (SMOTE)	BERT (doladěn), CLS	Random Forest	84,38%	63,21%	69,00%	80,36%	79,81%	79,40%	79,81%

Podobně jako u předchozího datasetu, i zde nedosahovala klasifikační hlava nejlepších výsledků. Výsledky klasifikační hlavy lze vidět v tabulce 17.

Tab. 17: Srovnání metod vyrovnání datasetu – klasifikační hlava modelu BERT (Endava)

			Makro průměr			Vážený průměr			
Metoda vyvážení	**Reprezentace**	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
Žádná	BERT (doladěn)	Klasifikační hlava	61,42%	40,62%	44,77%	76,72%	72,37%	70,59%	72,37%
Podvzorkování	BERT (doladěn)	Klasifikační hlava	57,58%	47,58%	51,08%	79,52%	80,34%	79,64%	80,34%
Odstranění min. kat.	BERT (doladěn)	Klasifikační hlava	72,18%	60,63%	64,70%	76,76%	77,19%	76,57%	77,19%
Augmentace	BERT (doladěn)	Klasifikační hlava	73,51%	65,29%	68,02%	78,14%	78,27%	78,07%	78,27%
Převzorkování	BERT (doladěn)	Klasifikační hlava	71,64%	62,25%	64,60%	78,41%	77,96%	77,68%	77,96%

V tabulce 18 jsou vyobrazeny nejlepší výsledky jednotlivých textových reprezentací nad augmentovaným datasetem. Je zde vidět podobná situace jako u datové

saty ALVAO. Přístupy využívající doladěný model BERT poskytují relativně dobré výsledky s tím, že přístup využívající CLS token doladěného modelu BERT a klasifikátor XGBoost poskytuje nejlepší výsledky. Přístupy, využívající nedoladěný model BERT, poskytují nejhorší F1 skóre, makro i vážené.

Tab. 18: Nejlepší výsledky pro všechny vektorové reprezentace – augmentovaný dataset (Endava)

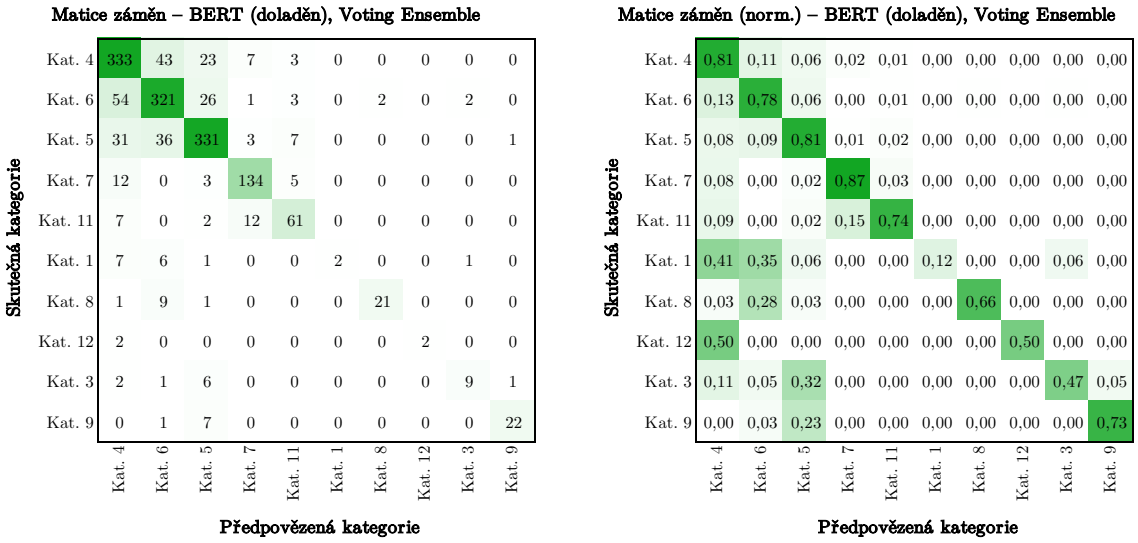
		Makro průměr			Vážený průměr			
Reprezentace	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
BERT (doladěn), CLS	XGBoost	79,22%	65,69%	70,19%	79,17%	79,23%	78,98%	79,23%
BERT (doladěn), CLS	Random Forest	79,27%	65,27%	69,97%	79,52%	79,55%	79,29%	79,55%
BERT (doladěn)	Klasifikační hlava	73,51%	65,29%	68,02%	78,14%	78,27%	78,07%	78,27%
BERT (doladěn), průměr	XGBoost	74,18%	63,21%	67,22%	76,95%	77,19%	76,94%	77,19%
TF-IDF	XGBoost	75,31%	58,69%	64,14%	75,98%	75,91%	75,48%	75,91%
FastText	XGBoost	78,83%	57,38%	63,04%	75,57%	75,08%	74,70%	75,08%
BERT (nedoladěn), CLS	XGBoost	77,51%	52,85%	58,82%	67,45%	65,88%	65,63%	65,88%
BERT (nedoladěn), průměr	KNN (4)	64,86%	53,60%	57,71%	64,75%	64,47%	64,32%	64,47%

Výsledky hyperparametrové optimalizace lze vidět v tabulce 19. Opět zde bylo dosaženo zanedbatelného zlepšení oproti nejlepším neoptimalizovaným výsledku. Podobně jako u prvního datasetu, i zde byly vybrány klasifikátory využívající využívající token CLS doladěného modelu BERT, nicméně tentokrát s algoritmy XGBoost, Random Forest a klasifikační hlava. Pro tento dataset dosahovalo sloučení optimalizovaných klasifikátorů do voting ensemble nejlepšího makro průměrovaného F1 skóre, na rozdíl od datasetu ALVAO.

Tab. 19: Výsledky optimalizace hyperparametrů a sloučení do voting ensemble – augmentovaný dataset (Endava)

		Makro průměr			Vážený průměr			
Reprezentace	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
BERT (doladěn), CLS a klas. hlava	Voting Ensemble	85,44%	65,03%	70,28%	79,45%	78,98%	78,66%	78,98%
BERT (doladěn), CLS	XGBoost	79,22%	65,69%	70,19%	79,17%	79,23%	78,98%	79,23%
BERT (doladěn), CLS	Random Forest	84,94%	64,89%	70,00%	79,14%	78,72%	78,40%	78,72%
BERT (doladěn)	Klasifikační hlava	85,30%	63,80%	69,25%	79,00%	78,34%	77,97%	78,34%

Jak je evidentní ze samotného F1 skóre, nejlepší klasifikátor stále produkuje poměrně velké množství chyb i přes všechny snahy o vyvážení a optimalizaci. Z matice záměn na obrázku 12 lze vyčíst, že má klasifikátor největší problémy s kategorií číslo 1, kde správně označil pouze 2 ze 17 vzorků. Dále má problémy s kategoriemi číslo 8, 12 a 3. I u kategorií s větším zastoupení dělá poměrně velké množství chyb a ani jednu kategorii nepřiřadil s 90% přesností.



Obr. 12: Matice záměn klasifikátoru voting ensemble s embeddingy BERT (doladěn)

4.2 Doporučení řešitele pro tiket

Jako u předchozích experimentů lze v tabulce 20 vidět efekt využití vyvažovacích metod na ALVAO datasetu pro doporučení řešitele. Byla zde skládána metoda odstranění menšinových kategorií a jedna z metod zvýšení zastoupení menšinových kategorií. Na rozdíl od předchozích experimentů je zde vidět jiný trend, kde jsou

nejlepší výsledky pro většinu metod poskytovány s reprezentací TF-IDF a klasifikátorem XGBoost. Opět nejlepší makro F1 skóre poskytovala metoda augmentace s textovou reprezentací pomocí CLS tokenu doladěného modelu BERT. Zajímavý poznatek je, že přístupy využívající TF-IDF mají často vyšší vážené F1 skóre než přístup s modelem BERT.

Tab. 20: Srovnání metod vyrovnání datasetu (doporučení řešitele)

			Makro průměr			Vážený průměr			
Metoda vyvážení	**Reprezentace**	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
Žádná	TF-IDF	XGBoost	41,28%	41,30%	41,02%	79,02%	79,96%	79,33%	79,96%
Podvzorkování	TF-IDF	XGBoost	41,44%	42,45%	41,74%	79,02%	79,76%	79,23%	79,76%
Odstranění min. kat.	TF-IDF	XGBoost	72,08%	72,43%	71,83%	80,95%	81,43%	80,98%	81,43%
Augmentace	BERT (doladěn), CLS	Logistic Regression	77,21%	74,15%	74,68%	79,81%	79,92%	79,73%	79,92%
Převzorkování	TF-IDF	XGBoost	73,43%	72,48%	72,51%	81,18%	81,74%	81,21%	81,74%
Převzorkování (SMOTE)	TF-IDF	XGBoost	73,90%	73,12%	73,10%	81,31%	81,74%	81,35%	81,74%

Jak je evidentní z tabulky 21, klasifikační hlava doladěného modelu BERT opět nedosahovala nejlepších výsledků s jakoukoliv metodou vyvážení. Dále lze z tabulky vidět, že algoritmický přístup využívající kosinovou podobnost poskytuje mnohem horší výsledky než přístupy využívající natrénované klasifikátory. Výsledek má dokonce o 1,7 % horší accuracy než doporučení dominantního řešitele pro danou kategorii a daný rok, což bylo zkoumáno v tabulce 10.

Tab. 21: Srovnání metod vyrovnání datasetu – klasifikační hlava modelu BERT (doporučení řešitele)

			Makro průměr			Vážený průměr			
Metoda vyvážení	**Reprezentace**	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
Žádná	Sentence Transformers	Algoritmus	31,01%	30,94%	30,48%	65,24%	63,73%	64,03%	63,73%
Žádná	BERT (doladěn)	Klasifikační hlava	36,86%	35,91%	36,17%	78,45%	80,06%	78,93%	80,06%
Podvzorkování	BERT (doladěn)	Klasifikační hlava	36,15%	37,00%	36,33%	78,97%	79,36%	78,91%	79,36%
Odstranění min. kat.	BERT (doladěn)	Klasifikační hlava	67,68%	67,78%	67,24%	80,48%	80,32%	80,08%	80,32%
Augmentace	BERT (doladěn)	Klasifikační hlava	72,15%	72,02%	71,24%	81,15%	81,13%	80,75%	81,13%
Převzorkování	BERT (doladěn)	Klasifikační hlava	69,36%	67,73%	68,10%	79,76%	80,42%	79,79%	80,42%

Nejlepší výsledky pro každou reprezentaci nad augmentovaným datasetem lze vidět v tabulce 22. Jak už bylo řečeno výše, nejlepší makro F1 skóre je zastoupeno textovou reprezentací CLS tokenem doladěného modelu BERT, nicméně přístup

využívající reprezentace pomocí TF-IDF a klasifikátor XGBoost není o tolik pozadu, a dokonce poskytuje lepší vážené F1 skóre. Nedoladěné varianty modelu BERT poskytují opět nejhorší výsledky v rámci tabulky.

Tab. 22: Nejlepší výsledky pro všechny vektorové reprezentace – augmentovaný dataset (doporučení řešitele)

		Makro průměr			Vážený průměr			
Reprezentace	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
BERT (doladěn), CLS	Logistic Regression	77,21%	74,15%	74,68%	79,81%	79,92%	79,73%	79,92%
TF-IDF	XGBoost	73,77%	73,53%	73,24%	81,83%	82,14%	81,79%	82,14%
BERT (doladěn)	Klasifikační hlava	72,15%	72,02%	71,24%	81,15%	81,13%	80,75%	81,13%
FastText	XGBoost	70,49%	69,01%	69,36%	77,72%	78,61%	77,87%	78,61%
BERT (doladěn), průměr	XGBoost	71,73%	67,74%	69,20%	80,16%	80,52%	80,26%	80,52%
BERT (nedoladěn), CLS	XGBoost	67,74%	62,67%	64,36%	75,09%	75,58%	74,94%	75,58%
BERT (nedoladěn), průměr	XGBoost	64,00%	64,25%	63,39%	75,17%	76,08%	75,14%	76,08%

Co se týče hyperparametrové optimalizace nejlepších klasifikátorů, tak výsledky těsně připomínají situaci z předchozího experimentu s ALVAO datasetem. Výsledky lze vidět v tabulce 23. Byly zde vybrány přístupy využívající reprezentace pomocí CLS tokenu doladěného modelu BERT a TF-IDF s algoritmy Logistic Regression, XGBoost a klasifikační hlava modelu BERT. Opět bylo dosaženo zanedbatelného zlepšení s tím, že sloučení vybraných klasifikátorů do voting ensemble neposkytovalo lepší výsledky než nejlepší optimalizovaný klasifikátor.

Tab. 23: Výsledky optimalizace hyperparametrů a sloučení do voting ensemble – augmentovaný dataset (doporučení řešitele)

		Makro průměr			Vážený průměr			
Reprezentace	**Klasifikátor**	**Precision**	**Recall**	**F1 skóre**	**Precision**	**Recall**	**F1 skóre**	**Accuracy**
BERT (doladěn), CLS	Logistic Regression	77,28%	74,20%	74,74%	79,87%	80,02%	79,81%	80,02%
TF-IDF	XGBoost	75,53%	73,38%	74,11%	81,89%	82,64%	81,99%	82,64%
BERT (doladěn), CLS a klas. hlava	Voting Ensemble	74,02%	73,35%	73,26%	82,12%	82,64%	82,16%	82,64%
BERT (doladěn)	Klasifikační hlava	72,15%	72,02%	71,24%	81,15%	81,13%	80,75%	81,13%

Výpočetní nároky vypadají vesměs podobně jako v prvním experimentu s ALVAO datasetem. Z tabulky 24 lze vidět, že voting ensemble opět poskytuje nejpomalejší dobu získání predikce pro 1 dokument a opět zde dělá hlavní zpomalení klasifikační hlava. Zajímavé srovnání je mezi přístupem využívající reprezentaci CLS tokenu doladěného modelu BERT s klasifikátorem Logistic Regression a přístupem využívající TF-IDF a XGBoost. Byť dosahuje druhý přístup o půl procenta nižší makro F1 skóre, jeho propustnost je téměř 140krát vyšší.

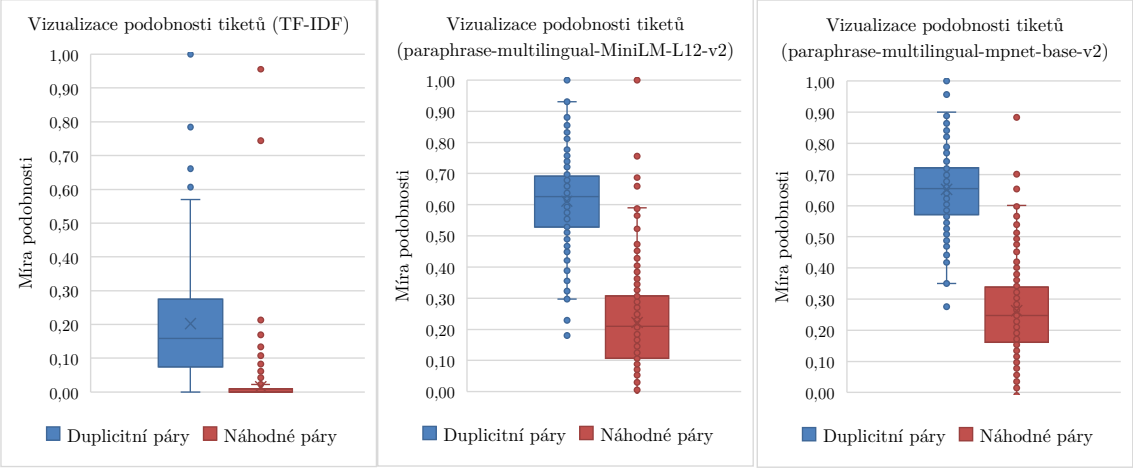
Tab. 24: Srovnání paměťových a časových nároků klasifikátorů (doporučení řešitele)

Reprezentace	Klasifikátor	F1 skóre (makro)	Velikost na disku [MB]	Čas načtení [ms]	Rychlost získání predikce pro 1 dokument [ms]	Maximální propustnost za 1 minutu [počet predikcí]
BERT (doladěn), CLS	Logistic Regression	74,74%	715,9	1617	365	164,4
BERT (doladěn), CLS a klas. hlava	Voting Ensemble	73,26%	731,6	3903	3016	19,9
BERT (doladěn)	Klasifikační hlava	71,24%	715,9	1668	2196	27,3
TF-IDF	XGBoost	74,11%	15,7	221	2,7	22813

4.3 Vyhledání a označení duplicitních tiketů

Na obrázku 14 jsou vyobrazeny krabicové grafy všech zkoumaných textových reprezentací pro vyhledání a označení duplicitních tiketů. Grafy zahrnují experiment využívající pouze prvních zpráv tiketů, nicméně grafy, zobrazující experiment využívající všech zpráv, vypadají velice podobně. Je zde vidět, že oba přístupy založené na předtrénovaných modelech Sentence Transformers mají velice podobné rozložení podobností duplicitních i náhodných párů s tím, že model paraphrase-multilingual-mpnet-base-v2 dosahuje mírně lepšího oddělení duplicitních a náhodných párů. Přístup využívající TF-IDF má mnohem nižší hodnoty podobnosti oproti předtrénovaným modelům, což samo o sobě nemusí být problém, nicméně spodní vous krabice duplicitních párů zcela zahrnuje krabici náhodných párů. Tento fakt může přinášet poměrně vysokou nejistotu při stanovení, zda je tiket k druhému duplicitní či nikoliv.

V tabulce 25 lze vidět výsledky experimentů pro vyhledání a označení duplicitních tiketů. Hodnoty pod sloupci „Prahová hranice“ a „Algoritmus“ představují procento označených duplicit ve vyhledaných kandidátech při testování přístupů nad všemi tikety, ke kterým je známý alespoň jeden označený duplicitní tiket. Z hodnot lze vidět, že všechny zkoumané přístupy dosahují poměrně nízkého podílu označených duplicit ve všech vyhledaných kandidátech – okolo 25 %. Relativně nízké hodnoty jsou v tomto experimentu očekávány, jelikož přístupy mnohdy vrací několik kandidátů duplicit pro zkoumaný tiket s tím, že ve většině případů mají zkoumané tikety pouze jeden označený duplicitní tiket. Tento jev může trochu lépe vyjasnit metrika recall, která udává, zda se k tiketům v kandidátech našly označené duplicitní tikety. Poskytnuté duplicity byly navíc označeny ručně, často s dalšími znalostmi, které nejsou obsaženy v rámci textu tiketu, jako například přiložený obrázek. Je tedy



(a) TF-IDF (b) MiniLM-L12-v2 (c) mpnet-base-v2

Obr. 14: Rozložení podobnosti duplicitních a náhodných párů (pouze první zpráva)

očekáváno, že používané metody pro vyhledání duplicitních tiketů nebudou schopny nalézt všechny označené duplicity.

Tab. 25: Srovnání přístupů pro vyhledání duplicit (podíl označených duplicit ve vyhledaných kandidátech a recall)

Model	**Počet zpráv**	**Prahová hranice (BERT 80 %, TF-IDF 20 %)**	**Algoritmus**	**Recall (prahová hranice)**	**Recall (Algoritmus)**
TF-IDF	První zpráva	10,07%	28,15%	40,20%	35,78%
TF-IDF	Všechny zprávy	12,82%	26,02%	37,25%	32,35%
paraphrase-multilingual-mpnet-base-v2	První zpráva	29,53%	28,06%	10,78%	27,94%
paraphrase-multilingual-mpnet-base-v2	Všechny zprávy	24,06%	21,45%	9,80%	18,14%
paraphrase-multilingual-MiniLM-L12-v2	První zpráva	32,91%	25,25%	10,78%	25,49%
paraphrase-multilingual-MiniLM-L12-v2	Všechny zprávy	18,18%	22,06%	4,90%	17,65%

Nejvyšší podíl označených duplicit ve vyhledaných kandidátech dosáhl přístup využívající předtrénovaný model paraphrase-multilingual-MiniLM-L12-v2 s využitím pouze první zprávy tiketu a metody prahové hranice. Téměř ve všech případech poskytuje využití pouze první zprávy lepší výsledky než zahrnutí všech zpráv od tvůrce tiketu. Výjimka tohoto pravidla je u reprezentace TF-IDF, kde se podíl označených duplicit zvýšil o 2,75 % při využívaní prahové hranice. Další zajímavost u přístupu TF-IDF je, že prahová hranice poskytuje mnohem horší výsledky než metoda využívající algoritmus. Relativně vysoká hodnota recall při tomto přístupu ukazuje, že bylo nalezeno velké množství kandidátů, ve kterých se snadněji

našly označené duplicity. U předtrénovaných modelů je tento jev obrácen a metoda prahové hranice poskytuje lepší výsledky ve většině případů. Nicméně recall je v těchto případech poměrně nízký, což naznačuje, že bylo vyhledáno poměrně malé množství kvalitnějších kandidátů. Metoda algoritmu dále dokázala v těchto případech podíl duplicit a recall vyrovnat. Nejlepší rovnováhu podílu označených duplicit ve vyhledaných kandidátech a hodnoty recall dosahuje přístup TF-IDF s metodou algoritmu.

Ačkoli tradiční přístup dosahoval nejlepších výsledků, ostatní přístupy nejsou o tolik pozadu a daly by se také využít, což posouvá rozhodnutí na jiné aspekty, jako například výpočetní nároky. Srovnání výpočetních nároků lze vidět v tabulce 26. Byť přístup TF-IDF zabírá nejméně místa na disku a má nejrychlejší inicializaci, poskytuje zdaleka nejnižší propustnost. Tento fakt se dá vysvětlit vysokým množstvím dimenzí výsledných vektorů tiketů, což značně zpomaluje výpočet kosinové podobnosti. Největší propustnosti zde dosahuje model paraphrase-multilingual-MiniLM-L12-v2, což odpovídá experimentům od Reimers (2022). Nicméně kde Reimers (2022) zpozoroval 4násobné zrychlení oproti paraphrase-multilingual-mpnet-base--v2, v tomto experiment byl pouze o zhruba 18 % rychlejší. Je zde tedy evidentní, že samotný výpočet kosinové podobnosti je nejsložitější překážka a zabírá největší množství času v rámci získání duplicit pro tiket.

Tab. 26: Srovnání paměťových a časových nároků přístupů pro vyhledání duplicit

Model	Počet zpráv	Velikost na disku [MB]	Čas načtení [ms]	Rychlost získání duplicit pro 1 tiket [ms]	Maximální propustnost za 1 minutu [počet predikcí]
TF-IDF	První zpráva	7,8	5	1366	44
TF-IDF	Všechny zprávy	13,1	8,7	1429	42
paraphrase-multilingual-mpnet-base-v2	První zpráva	995	3810	613	97,8
paraphrase-multilingual-mpnet-base-v2	Všechny zprávy	998,5	3810	716	83,8
paraphrase-multilingual-MiniLM-L12-v2	První zpráva	434,7	2165	517	116
paraphrase-multilingual-MiniLM-L12-v2	Všechny zprávy	438,2	2165	554	108,3

4.4 Optimalizace a nasazení

Pro nasazení byly zvoleny následující přístupy:

- Přiřažení kategorie k tiketu
  - Reprezentace – Slavic BERT (doladěn), CLS (augmentovaný dataset)
  - Klasifikátor – XGBoost
- Doporučení řešitele pro tiket
  - Reprezentace – TF-IDF (augmentovaný dataset)

  - Klasifikátor – XGBoost
- Vyhledání a označení duplicitních tiketů
  - Reprezentace – paraphrase-multilingual-MiniLM-L12-v2
  - Přístup – algoritmus

V případě přiřazení kategorie k tiketu jedná o nejlepší přístup. Pro doporučení řešitele pro tiket byl vybrán druhý nejlepší přístup, využívající TF-IDF a XGBoost, jelikož dosahoval velice podobného makro F1 skóre jako nejlepší přístup, a navíc dosahoval lepšího váženého F1 skóre a téměř 140krát vyšší propustnosti. Pro vyhledání a označení duplicitních tiketů byl zvolen přístup, který poskytoval nejvyšší propustnost. Před nasazením byly modely BERT optimalizovány pomocí frameworku ONNX. Výsledky optimalizace lze vidět v tabulce 27. V případě predikce kategorie lze vidět drastické zrychlení už jenom při převedení modelu do ONNX formátu, kde se propustnost zvýšila o 10násobek. Kvantizace poté dále poskytuje další zrychlení o přibližně 22 %. V případě získání duplicitních tiketů je zrychlení mnohem nižší. Samotný převod do ONNX formátu zrychlil získání predikce o 16 % a kvantizace poskytnula další 2 % zrychlení. Opět je zde vidět, že výpočet kosinové podobnosti zabírá největší podíl času získání predikce.

Tab. 27: Srovnání optimalizace modelů typu BERT pro nasazení

Úkol	Model	Formát	Velikost na disku [MB]	Čas načtení [ms]	Rychlost získání predikce pro 1 tiket [ms]	Maximální propustnost za 1 minutu [počet predikcí]
Predikce kategorie	Slavic BERT (doladěný)	PyTorch	716	1663	500	120
		ONNX	716	394	46,7	1284
		ONNX – Kvantizace	178,3	195	38,2	1570
Získání duplicitních tiketů	paraphrase-multilingual-MiniLM-L12-v2	PyTorch	434,7	2165	517	116
		ONNX	434,7	648	446	134
		ONNX – Kvantizace	132,7	530	437	137

V tabulce 28 je poté vidět typické doby obsloužení 1 požadavku konečné nasazené služby v rámci Azure na B2 App Service plánu. Služba byla napojena na testovací databázi obsahující celkem 3419 tiketů, které bylo nejdříve nutné vektorizovat a uložit do pomocné struktury pro správný chod úkolu získání duplicitních tiketů ke vstupnímu tiketu. Hodnoty v tabulce zahrnují veškerou režii spojenou s posíláním požadavku, přijímáním požadavku na službě, samotným výpočtem, odesíláním odpovědi a přijetím odpovědi. Je zde vidět, že se doba obsloužení obecně odvíjí od rychlosti získání predikce použitých modelů. Úkol doporučení řešitele využívá přístup TF-IDF a XGBoost, což dosahovalo velice vysoké rychlosti získání predikce, a i zde je ze třech zkoumaných úkolů nejrychlejší. Podobně tak je úkol získání duplicitních tiketů nejpomalejší, jelikož trval nejdéle před nasazením. Pokud srovnáme

rychlosti získání predikce před nasazením a po nasazení tak lze zjistit, že režie služby a komunikace přidává v průměru 240,7 ms do celkové doby obsloužení.

Tab. 28: Doba obsloužení požadavku nasazené služby

Úkol	**Doba obsloužení 1 požadavku [ms]**
Predikce kategorie	312
Doporučení řešitele	224
Získání duplicitních tiketů	664

5 Diskuze

V rámci této práce byly provedeny tři hlavní experimenty, z kterých bylo zjištěno několik zajímavých poznatků. Byly zde porovnány různé přístupy reprezentace textových dat, techniky vyvážení datové sady a algoritmy pro uskutečnění klasifikace.

5.1 Poznatky z kategorizace tiketů

První experiment, zabývající se přiřazením kategorie k nově vytvořenému tiketu, ukázal, že moderní kontextové textové reprezentace dokážou poskytnout mnohem lepší výsledky než tradiční přístupy. Toto zlepšení se především projevilo v makro průměrovaném F1 skóre. Zároveň zde bylo zjištěno, že pro efektivní využití embeddingů získaných pomocí modelu BERT je nutné model nejdříve doladit na daném úkolu. I přesto, že výsledná klasifikační hlava z doladění neposkytovala nejlepší výsledky, embeddingy posledního kodéru tohoto modelu byly klíčové pro dosažení nejlepších výsledků.

Dále bylo také zjištěno, že vyvážení datasetu mnohdy pomáhá pro zvýšení kvality výsledného klasifikátoru. U datové sady ALVAO měla největší dopad augmentace datasetu pomocí strojového překladu s tím, že ostatní zkoumané techniky v tomto případě mírně zhoršili výsledky. Naopak u datové sady Endava měly vyvažující techniky mnohem větší dopad a bez nich by byl výsledný klasifikátor prakticky nepoužitelný. Ze třech technik zvýšení reprezentace menšinových kategorií byla v obou případech nejefektivnější technika augmentace pomocí strojového překladu.

Při zkoumání efektu optimalizace hyperparametrů nejlepších klasifikátorů bylo zjištěno, že je tento proces mnohdy přebytečný. Byť bylo pro oba datasety dosaženo lepších výsledků než s neoptimalizovanými klasifikátory, rozdíl ve zkoumaných metrikách byl vždy prakticky zanedbatelný. Podobně tak dopadly experimenty se sloučením nejlepších klasifikátorů do voting ensemble, kde byla navíc mnohem zpomalena rychlost predikce. V případě optimalizace hyperparametrů modelu BERT byl tento proces navíc velmi časově náročný – v průměru trvalo 15 iterací s různými hyperparametry okolo 6 hodin na GPU NVIDIA Tesla T4, přičemž po 6 hodinách byl dosažen limit na službě Google Colab, který GPU odstavil na přibližně 2 dny.

Co se týče samotných hodnot nejlepších přístupů, tak bylo u ALVAO datasetu dosaženo makro F1 skóre rovno 86,94 %, váženého F1 skóre rovno 97,60 % a accuracy 97,61 %. Pro Endava dataset bylo dosaženo 70,28 % makro F1 skóre, 78,66 % váženého F1 skóre a accuracy 78,98 %. Pro ALVAO dataset jsou tyto výsledky poměrně dobré, zejména z váženého hlediska, a hlavní problémy zde dělá pouze 1 menšinová kategorie, která je velice podobná jiné. Pokud by byly tyto 2 podobné kategorie sloučeny, tak by bylo možné dosáhnout mnohem vyššího makro F1 skóre.

Výsledky s Endava datasetem jsou naopak mnohem nižší a je obtížné říct, zda by byl takový klasifikátor užitečný v praxi. Jelikož je datová sada anonymizována, tak je obtížné říct, kde byl hlavní problém, zda byly problematické kategorie velice

podobné, jako u datasetu ALVAO, či zda byl problém jinde. Vstupní dataset Endava byl velice nevyvážený, mnohem více než dataset ALVAO, což ani vyvažovací techniky nebyly schopny plně vyřešit. Textová data zde navíc byla poskytnuta už agresivně předzpracována, což mohlo snížit efektivitu kontextových embeddingů. Nicméně i přes toto předzpracování poskytovaly kontextové embeddingy nejlepší výsledky. Byť jsou výsledné hodnoty metrik poměrně nízké, jsou pořád mnohem vyšší než výsledky Żak et al. (2021), kteří tento dataset poskytnuli. Żak et al. navíc odstranili o 2 další menšinové kategorie více, než bylo odstraněno v této diplomové práci, a dokázali tak vybudovat klasifikátor dosahující pouze 31,26 % makro F1 skóre a 79 % váženého F1 skóre. Autoři si nízké výsledky vysvětlují extrémní mírou nevyváženosti, což by z velké části výsledky diplomové práce podporovaly, nicméně stále to nevysvětluje relativně slabý výkon klasifikátoru i u kategorií s velkým zastoupením. Jak už bylo řečeno, bez bližších znalostí neanonymizovaného datasetu je obtížné stanovit příčinu slabého výkonu u většinových kategorií.

Ačkoli není možné výsledky přímo srovnávat s ostatními pracemi již na základě odlišných datasetů, srovnání může i tak poskytnout zajímavé informace. Například pokud srovnáme výsledky datasetu ALVAO s prací využívající tradiční přístupy od Eichhorn (2020), kde obě datové sady obsahují 8 kategorií, tak můžeme vidět, že nejlepší přístup zkoumaný v diplomové práci dosahuje o 12,61 % vyšší accuracy. Obecně se hodnoty accuracy nejlepších výsledků obou zkoumaných datových sad pohybují ve stejné oblasti, jako již existující práce zabývající se podobnou problematikou.

5.2 Poznatky z doporučení řešitele

Již předběžné testování druhého experiment, zabývající se doporučením řešitele pro tiket, ukázalo, že doporučení či přiřazení pouze na základě textu je prakticky nepoužitelné. Z extensivnějších experimentů bylo poté zjištěno, že nadvláda modelů založeným na transformeru není úplná, jak by bylo naznačeno v prvním experimentu. Opět se zde osvědčily vyvažovací metody, zejména odstranění menšinových kategorií a augmentace. Byť zde bylo opět dosaženo nejvyššího makro průměrovaného F1 skóre pomocí embeddingů doladěného modelu BERT, tradiční přístupy založené na TF-IDF nebyly o tolik horší, a dokonce dosahovaly vyššího váženého F1 skóre. Po optimalizaci hyperparametrů byl rozdíl v makro F1 skóre pouze 0,63 % ve prospěch moderního přístupu, nicméně vážené F1 skóre bylo o 2,18 % vyšší u tradičního přístupu. Z těchto poznatků lze tedy konstatovat, že při zapojení netextových atributů do klasifikace je vhodné prozkoumat i tradiční přístupy. Další výhoda tradičního přístupu v tomto experimentu je drasticky vyšší propustnost algoritmu, což může být klíčové pro finální nasazení.

Samotné hodnoty nejlepších výsledků druhého experimentu jsou relativně nízké ve srovnání s výsledky prvního experimentu s ALVAO datasetem. Klasifikátor založený na TF-IDF a XGBoost zajišťuje nejlepší rovnováhu makro F1 skóre, váženého F1 skóre a propustnosti, nicméně stále dělá poměrně velké množství chyb. Ve srovnání s doporučením dominantního řešitele na základě kategorie a roku dosahuje

o 17,18 % vyšší accuracy. Možné zefektivnění tohoto klasifikátoru by mohlo spočívat v doporučení dvou nebo tří nejvhodnějších řešitelů, podobně jako zkoušel Opuchlich (2019) ve své práci. Operátor by z těchto řešitelů poté mohl vybrat nejvhodnějšího například na základě počtu aktuálně řešených úkolů každého řešitele. Tento proces by bylo možné dále automatizovat.

5.3 Poznatky z hledání duplicitních tiketů

Třetí experiment, zabývající se vyhledáním a označením duplicitních tiketů, ukázal, že všechny zkoumané reprezentace poskytovaly poměrně nízký podíl označených duplicit ve vyhledaných kandidátech. Obecně bylo u všech přístupů obtížné rozlišit duplicitní tiket od velice podobného. Obě zkoumané metody vyhledání duplicit poskytovaly ve většině případů podobné podíly označených duplicit ve vyhledaných kandidátech, nicméně metoda algoritmu dokázala značně zvýšit metriku recall u předtrénovaných modelů. Hlavní výjimkou byla reprezentace TF-IDF, kde byla metoda algoritmu mnohem lepší než prahová hranice. Nejlepší výsledky všech reprezentací se pohybovaly okolo 25 % označených duplicit ve vyhledaných kandidátech.

Tradiční přístup využívající TF-IDF dosahoval nejvyšších hodnot metriky recall. S využitím metody algoritmu poté tento přístup dosahoval nejlepší rovnováhy mezi podílem označených duplicit ve vyhledaných kandidátech a recall. Nicméně při zkoumání výpočetních nároků bylo zjištěno, že vysoký počet dimenzí reprezentace TF-IDF značně snižoval maximální propustnost oproti modelům založeným na modelu BERT. Vysoké množství dimenzí tradičního přístupu může příliš zpomalovat získání duplicitních tiketů, například při zakládání nového tiketu, zejména pokud je zakládáno několik tiketů ve stejnou dobu, což jej může dělat zcela nevhodný pro produkční nasazení. Na základě konečné implementace řešení mohou tedy být nízké výpočetní nároky mnohem více důležité než malé zlepšení kvality výsledků, a proto je vhodné se zaměřit na modernější řešení.

5.4 Přenesení výsledků experimentů do provozu

Co se týče nasazení nejlepších řešení z výše uvedených experimentů, tak byla optimalizace vstupních modelů klíčová ve snížení doby odezvy. Nejvhodnější řešení dvou ze tří experimentů používalo model odvozený od BERTa, který je ve své neoptimalizované formě poměrně náročný na výpočetní prostředky, zejména pokud je využito CPU inference. Optimalizace těchto modelu pomocí frameworku ONNX přineslo v prvním případě drastické zrychlení získání predikce o 10násobek. V druhém případě nebylo zrychlení tak velké, jelikož nejpomalejší část zůstala beze změny, nicméně určitého zrychlení bylo dosaženo i zde. Ačkoli kvantizace přinesla další zrychlení, její dopad oproti standardnímu ONNX modelu nebyl tak velký. Rozhodnutí, zda využít kvantizaci či ne, by tedy spočívalo spíše v zachování identických výstupů jako neoptimalizovaný model.

Výsledná řešení s korespondujícími modely byla poté nasazena na Azure App Service s plánem B2. Roční náklady takovéto služby by činily podle odhadu zhruba 258,6 EUR. Jedná se o CPU inferenci, což výrazně snižuje výsledné náklady. Pokud by služba využívala rychlejší GPU inferenci, tak by se náklady drasticky zvýšily na odhadovaných 4 300,11 EUR za rok při využití nejlevnějšího GPU virtuální stroje na AKS. Nasazení v rámci této práce bylo stavěno pouze na testovací účely a ověření konceptu, je proto velice možné, že pro produkční nasazení by plán B2 nestačil a bylo by nutné použít dražší řešení, což by záleželo na zátěžových testech služby.

6 Závěr

Tato práce se zabývala vybudováním systému pro automatickou kategorizaci zákaznických požadavků. Pro uskutečnění tohoto cíle bylo využito několika metod z oblasti zpracování přirozeného jazyka. Nejdříve byla prozkoumána oblast moderní zákaznické podpory, její struktura, časté problémy a důležité faktory. Následně byla prozkoumána samotná oblast zpracování přirozeného jazyka včetně posledních průlomů. Poté byla provedena analýza existujících prací zabývající se stejnou problematikou. Dále byly analyzovány možnosti aplikace metod a procesů zpracování přirozeného jazyka z hlediska odezvy a nákladů.

Ze zjištěných poznatků byly vytvořeny tři hlavní experimenty, každý zabývající se důležitou částí procesu roztřídění a kategorizace zákaznických požadavků. Přístupy poskytující nejlepší výsledky v daném experimentu byly poté adaptovány do služby, která byla úspěšně nasazena na cloudovou platformu Microsoft Azure.

Experiment přiražení kategorie k tiketu na základě jeho textu byl prováděn na dvou datových sadách. Bylo zjištěno, že moderní přístupy využívající textové reprezentace získané pomocí doladěného modelu BERT poskytují nejlepší základ pro vytvoření kvalitního klasifikátoru. Dále bylo zjištěno, že techniky pro vyvážení datasetu jsou pro data typu zákaznických požadavků velice důležité, v některých případech nutné, a mnohdy dokážou zvýšit kvalitu výsledného klasifikátoru. Z metod zvýšení reprezentace menšinových kategorií poskytovala nejlepší výsledky augmentace pomocí strojového překladu.

Druhý experiment, zabývající se doporučením řešitele, ukázal, že kvalitní doporučení na základě samotného textu tiketu je prakticky nemožné. Opět se zde osvědčily metody pro vyvážení datasetu. Dále zde bylo zjištěno, že při kombinaci netextových atributů s vektorizovaným textem dosahují tradiční přístupy velmi dobrých výsledků ve srovnáním s moderním řešení.

Poslední experiment zdůraznil využití moderních přístupů při vyhledání a označení duplicitních tiketů. Bylo zjištěno, že tradiční přístup je příliš zpomalován výpočtem kosinové podobnosti, což jej dělá nevhodným pro finální nasazení při vyhledávání duplicit pro nově vytvořený tiket.

Možné rozšíření této práce by mohlo spočívat v prozkoumání vlivu dalších předtrénovaných modelů na výsledky klasifikace. Další cesta rozšíření by mohla zkoumat efekt aktivního učení pro zmírnění problému nevyváženého datasetu u prvního a druhého experimentu. Pro druhý experiment by mohlo rozšíření obnášet zapojení více netextových atributů do klasifikačního procesu, jako například počet aktivně řešených úkolů jednotlivých řešitelů. Pro třetí experiment by mohl být zkoumán efekt doladění modelu. Byť bylo doladění modelů pro vyhledání duplicit v této práci nemožné z důvodu nedostatku označených dat, z prvních dvou experimentů se ukázalo, že doladěný model BERT poskytuje mnohem lepší výsledky než pouze předtrénovaný.

Jak už bylo řečeno, cílem práce bylo vybudování automatizovaného systému

pro kategorizaci zákaznických požadavků psaných přirozeným jazykem. Z výše uvedených odstavců lze konstatovat, že cíl byl úspěšně naplněn. Veškeré skripty využité pro realizaci experimentů jsou k dispozici v elektronické příloze včetně vybudované služby, která je připravena pro nasazení na Azure.

7 Literatura

AGGARWAL, C. C. *Machine Learning for Text* New York: Springer, 2018, s. 237-238. ISBN 978-3-319-73531-3.

AGGARWAL, C. C. *Data Classification: Algorithms And Applications* Boca Raton: Chapman and Hall/CRC, 2020, s. 452-453. ISBN 978-0-367-65914-1.

ALAMMAR J. *The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.* [online] Jay Alammar, 2018. [cit. 26. 2. 2022]. Dostupné z: https://jalammar.github.io/illustrated-transformer/.

ARKHIPOV, M. ET AL. *Tuning Multilingual Transformers for Language-Specific Named Entity Recognition* Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing, 2019, 89-93. Dostupné z: doi: 10.18653/v1/W19-3712.

BEISSE, F. *A Guide to Computer User Support for Help Desk and Support Specialists* Boston, Cengage Learning, 2014, s. 241-242. 978-1-28-585268-3.

BENGFORT, B. ET AL. *Applied Text Analysis with Python: Enabling Language-Aware Data Products with Machine Learning* California: O'Reilly Media, 2018, s. 14. ISBN 978-1-49-196304-3.

BRIGGS, J. *Sentiment Analysis With Long Sequences | Towards Data Science* [online] Towards Data Science, 2021a. [cit. 1. 3. 2022]. Dostupné z: https://towardsdatascience.com/how-to-apply-transformers-to-any-length-of--text-a5601410af7f.

BRIGGS, J. *How to Fine-Tune BERT Transformer Python | Towards Data Science* [online] Towards Data Science, 2021b. [cit. 1. 3. 2022]. Dostupné z: https://towardsdatascience.com/how-to-train-bert-aaad00533168.

BROWNLEE, J. *How to Develop Voting Ensembles With Python* [online] Machine Learning Mastery, 2020. [cit. 22. 2. 2022]. Dostupné z: https://machinelearningmastery.com/voting-ensembles-with-python/.

BROWNLEE, J. *What is a Confusion Matrix in Machine Learning* [online] Machine Learning Mastery, 2016. [cit. 20. 2. 2022]. Dostupné z: https://machinelearningmastery.com/confusion-matrix-machine-learning/.

CHOI, H. ET AL. *Evaluation of BERT and ALBERT Sentence Embedding Performance on Downstream NLP Tasks* [online], 2021. [cit. 7. 4. 2022]. Dostupné z: https://arxiv.org/abs/2101.10642.

COULOMBE, C. *Text Data Augmentation Made Simple By Leveraging NLP Cloud APIs* [online], 2018. [cit. 1. 5. 2022]. Dostupné z: https://arxiv.org/abs/1812.04718.

DEBUT, L. *Benchmarking Transformers: PyTorch and TensorFlow | by Lysandre Debut | HuggingFace | Medium* [online] Meidum, 2019. [cit. 28. 2. 2022]. Dostupné z: https://medium.com/huggingface/benchmarking-transformers--pytorch-and-tensorflow-e2917fb891c2.

DEVLIN, J. ET AL. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* [online] abs/1706.03762, 2018. [cit. 1. 3. 2022]. Dostupné z: https://arxiv.org/abs/1810.04805.

EICHHORN, G. *Predict IT Support tikets with Machine Learning and NLP | by Garrett Eichhorn | Towards Data Science* [online] Towards Data Science, 2020. [cit. 22. 2. 2022]. Dostupné z: https://towardsdatascience.com/predict-it--support-tikets-with-machine-learning-and-nlp-a87ee1cb66fc.

FELDMAN, R., SANGER, J. *Machine Learning for Text.* New York: Cambridge University Press, 2007, s. 1-81. ISBN 0-521-83657-3.

GANEGEDARA, T. *Natural Language Processing with TensorFlow* Birmingham: Packt Publishing, 2018, s. 67. ISBN 978-1-78-847831-1.

GARG, A. *Why use Azure Functions for ML inference ? - Microsoft Tech Community* [online] Microsoft Tech Community, 2020. [cit. 25. 2. 2022]. Dostupné z: https://techcommunity.microsoft.com/t5/apps-on-azure--blog/why-use-azure-functions-for-ml-inference/ba-p/1416728.

GOLDBERG, Y. *Machine Learning for Text* San Rafael: Morgan & Claypool Publishers, 2017, s. 1. ISBN 978-1-62705-298-6..

GU, K., BUDHKAR, A. *A Package for Learning on Tabular and Text Data with Transformers* In Proceedings of the Third Workshop on Multimodal Artificial Intelligence. 2021,69-73. Dostupné z: 10.18653/v1/2021.maiworkshop-1.10.

GU, Q. ET AL. *Evaluation Measures of the Classification Performance of Imbalanced Data Sets* Computational Intelligence and Intelligent Systems. ISICA 2009 (51), 2009, 461-471. Dostupné z: doi: 10.1007/978-3-642-04962-0_53.

HASHESH, A. *Compressive Transformer vs LSTM. a summary of the long term memory… | by Ahmed Hashesh | DS4B | Medium* [online] Medium, 2020. [cit. 17. 2. 2022]. Dostupné z: https://medium.com/ml2b/introduction-to--compressive-transform-53acb767361e.

HUGGING FACE *Exporting transformers models — transformers 3.3.0 documentation* [online] Hugging Face, 2020. [cit. 13. 3. 2022]. Dostupné z: https://huggingface.co/transformers/v3.3.1/serialization.html.

HUGGING FACE *Transformer models - Hugging Face Course* [online] Hugging Face, 2022a. [cit. 26. 2. 2022]. Dostupné z:

https://huggingface.co/course/chapter1/4?fw=pt.

Hugging Face *Exporting Hugging Face Transformers Models* [online] Hugging Face, 2022b. [cit. 28. 2. 2022]. Dostupné z: https://huggingface.co/docs/transformers/serialization#onnx.

Kerner, S. M. *What is autoscaling? Cloud autoscaling explained* [online] SearchEnterpriseCloudComputing, 2021. [cit. 28. 2. 2022]. Dostupné z: https://www.techtarget.com/searchcloudcomputing/definition/autoscaling.

KPMG *Enterprise reboot* [online] KPMG, 2020. [cit. 2. 2. 2022]. Dostupné z:https://info.kpmg.us/content/dam/advisory/en/pdfs/2020/enterprise--reboot.pdf.

Landsman, I. *A Guide to Support tiket Categorization* [online] Helpspot, 2015. [cit. 6. 3. 2022]. Dostupné z: https://www.helpspot.com/blog/a-guide-to-support-tiket-categorization.

Lee, J. et al. *What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning* [online] 2019. [cit. 25. 3. 2022]. Dostupné z: https://arxiv.org/abs/1911.03090.

Lewinson, E. *Python for Finance Cookbook* Birmingham: Packt Publishing, 2020, s. 269. ISBN 978-1-78961-851-8.

Li, Y. *Faster and smaller quantized NLP with Hugging Face and ONNX Runtime | by Yufeng Li | Microsoft Azure | Medium* [online] Medium, 2020. [cit. 28. 2. 2022]. Dostupné z: https://medium.com/microsoftazure/faster-and--smaller-quantized-nlp-with-hugging-face-and-onnx-runtime-ec5525473bb7.

Liu, Q. et al. *A Survey on Contextual Embeddings* [online], 2020. [cit. 15. 2. 2022]. Dostupné z: https://arxiv.org/abs/2003.07278.

Liu, Y. et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach* [online], 2019. [cit. 1. 3. 2022]. Dostupné z: https://arxiv.org/abs/1907.11692.

Martin, J. *What is a support tiket? How service tikets work* [online] Zendesk, 2019. [cit. 11. 3. 2022]. Dostupné z: https://www.zendesk.com/blog/what-is-a-support-tiket/.

Menken, I., Blokdijk, G. *Support Center Complete Handbook* Brisbane: Emereo Publishing, 2009, s. 68. 978-1-74-244130-6.

McCormick, C. *BERT Word Embeddings Tutorial · Chris McCormick* [online] Chris McCormick, 2019. [cit. 1. 3. 2022]. Dostupné z: https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/.

Mikolov, T. et al. *Efficient Estimation of Word Representations in Vector Space* Proceedings of Workshop at ICLR [online], 2013. [cit. 25. 2. 2022].

Dostupné z: https://arxiv.org/abs/1301.3781.

Minaee, S. et al. *Deep Learning Based Text Classification: A Comprehensive Review* [online] 2021. [cit. 7. 3. 2022]. Dostupné z: https://arxiv.org/abs/2004.03705.

Miner, G. *Practical text mining and statistical analysis for non-structured text data applications* Waltham, MA: Academic Press, 2012, s. 46-50. ISBN 978-0-12-386979-1.

Microsoft *Deploy machine learning models - Azure Machine Learning | Microsoft Docs* [online] Microsoft Docs, 2022a. [cit. 28. 2. 2022]. Dostupné z: https://docs.microsoft.com/en-us/azure/machine-learning/how-to-deploy-and--where?tabs=azcli.

Microsoft *Overview - Azure App Service | Microsoft Docs* [online] Microsoft Docs, 2022b. [cit. 28. 2. 2022]. Dostupné z: https://docs.microsoft.com/en-us/azure/app-service/overview.

Microsoft *Machine learning inference during deployment - Cloud Adoption Framework | Microsoft Docs* [online] Microsoft Docs, 2021. [cit. 25. 2. 2022]. Dostupné z: https://docs.microsoft.com/en-us/azure/cloud-adoption--framework/innovate/best-practices/ml-deployment-inference.

Olson, S. *10 help desk metrics for service desks and internal help desks* [online] Zendesk, 2018. [cit. 6. 3. 2022]. Dostupné z: https://www.zendesk.com/blog/top-10-help-desk-metrics/.

Opuchlich, P. *Text Classification of Support tiket Data of SAP* [online] ResearchGate, 2019. [cit. 23. 2. 2022]. Dostupné z: https://www.researchgate.net/publication/340492353_Text_Classification_of_Support_tiket_Data_of_SAP.

Patruno, L. *Batch Inference vs Online Inference - ML in Production* [online] ML in Production, 2019. [cit. 25. 2. 2022]. Dostupné z: https://mlinproduction.com/batch-inference-vs-online-inference/.

Paramesh, S. P. et al. *Classifying the Unstructured IT Service Desk tikets Using Ensemble of Classifiers.* 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS), 2018, 221-227. Dostupné z: doi:10.1109/CSITSS.2018.8768734.

Parmar, P. S et al. *Multiclass Text Classification and Analytics for Improving Customer Support Response through different Classifiers.* 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2018, 538-542. Dostupné z: doi:10.1109/ICACCI.2018.8554881.

Peters, M. E. et al. *Deep Contextualized Word Representations* In:

Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers) [online]. Stroudsburg, PA, USA: Association for Computational Linguistics, 2018, 2018, s. 2227-2237 [cit. 2022-02-17]. Dostupné z: doi:10.18653/v1/N18-1202.

Pilehvar, M. T., Camacho-Collados, J. *Embeddings in Natural Language Processing: Theory and Advances in Vector Representations of Meaning.* San Rafael: Morgan & Claypool, 2020. ISBN 978-1-636-39022-2.

Prabhu, S. et al. *Multi-class Text Classification using BERT-based Active Learning* [online] 2021. [cit. 7. 3. 2022]. Dostupné z: https://arxiv.org/abs/2104.14289.

Raschka, S. *Python Machine Learning* 1. Birmingham: Packt Publishing, 2015, s. 190-192. ISBN 978-1-78-355513-0.

Reimers, N., Gurevych, I. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks* [online], 2019. [cit. 1. 3. 2022]. Dostupné z: https://arxiv.org/abs/1908.10084.

Reimers, N. *Pretrained Models — Sentence-Transformers documentation* [online], 2022. [cit. 30. 3. 2022]. Dostupné z: https://www.sbert.net/docs/pretrained_models.html#model-overview.

Rogers, A. et al. *A Primer in BERTology: What We Know About How BERT Works* Transactions of the Association for Computational Linguistics, 2020, 842-866. Dostupné z: doi:10.1162/tacl_a_00349.

Sanh, V. et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter* [online], 2019. [cit. 1. 3. 2022]. Dostupné z: https://arxiv.org/abs/1910.01108.

Sammut, C., Webb, G. I. *Encyclopedia of Machine Learning* New York: Springer, 2010, s. 10-293. ISBN 978-0-387-34558-1.

Sapunov, G. *Speeding up BERT. How to make BERT models faster | by Grigory Sapunov | Intento* [online] Intento, 2019. [cit. 1. 3. 2022]. Dostupné z: https://blog.inten.to/speeding-up-bert-5528e18bb4ea.

Scikit-Learn *sklearn.metrics.classification_report — scikit-learn 1.0.2 documentation* [online] Scikit-learn, 2022. [cit. 5. 3. 2022]. Dostupné z: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html.

Sokolova, M., Lapalme, G. *A systematic analysis of performance measures for classification tasks* Information Processing & Management, 45 (4), 2009, 427-437. ISSN 0306-4573. Dostupné z: doi/10.1016/j.ipm.2009.03.002.

Sun, C. et al. *How to Fine-Tune BERT for Text Classification?* [online], 2019. [cit. 1. 5. 2022]. Dostupné z: https://arxiv.org/abs/1905.05583.

Taylor, J. *ELMo: Contextual language embedding | by Josh Taylor | Towards Data Science* [online] Towards Data Science, 2019. [cit. 17. 2. 2022]. Dostupné z: https://towardsdatascience.com/elmo-contextual-language-embedding--335de2268604.

Vaswani, A. et al. *Attention Is All You Need* [online] abs/1706.03762, 2017. [cit. 26. 2. 2022]. Dostupné z: http://arxiv.org/abs/1706.03762.

Żak, K et al. *GitHub - karolzak/support-tickets-classification* [online] GitHub, 2021. [cit. 28. 3. 2022]. Dostupné z: https://github.com/karolzak/support-tickets-classification.

Zhong, J., Li, W. *Predicting Customer Call Intent for the Auto Dealership Industry from Analyzing Phone Call Transcripts With CNN for Multiclass Classification.* International Journal on Soft Computing, Artificial Intelligence and Applications. 8. 2019,13-25. Dostupné z: doi:10.5121/ijscai.2019.8302.

Zhou, Z. *Ensemble Methods: Foundations and Algorithms.* Boca Raton: Chapman and Hall/CRC, 2012. ISBN 978-1-4398-3005-5.

Zhu, Y. et al. *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books* Proceedings of the IEEE international conference on computer vision, 2015, 19-27. Dostupné z: doi:10.1109/ICCV.2015.11.

Žižka, J. et al. *Text mining with machine learning: principles and techniques.* Boca Raton: CRC Press, 2020, s. 142. ISBN 978-1-138-60182-6.

Seznam obrázků

Seznam tabulek

Přílohy

A Obsah CD

Přiložené CD obsahuje následující položky:

- Adresář *zdrojovy kod* obsahující zdrojový kód ke všem prováděným experimentům.
- Adresář *vysledky* obsahující veškeré výsledky a vizualizace všech dílčích experimentů v rámci třech hlavních experimentů.
- Adresář *sluzba* obsahující zdrojový kód vytvořené služby. Kód je připraven na sestavení a nasazení na Microsoft Azure.

B Hyperparametry optimalizovaných modelů

B.1 Přiřazení kategorie k tiketu

B.1.1 Datová sada ALVAO

BERT (doladěn), CLS – XGBoost

- max_depth – 3
- learning_rate – 0,2
- subsample – 0,9
- colsample_bytree – 0,8
- colsample_bylevel – 0,9
- n_estimators – 150

BERT (doladěn), CLS – KNN

- K – 3

BERT (doladěn) – Klasifikační hlava

- layer_0-2 – 0,00004
- layer_2-4 – 0,00004
- layer_4-6 – 0,00004
- layer_6-8 – 0,00004
- layer_8-10 – 0,00004
- layer_10-12 – 0,00004
- bert.pooler.dense.bias – 0,00004
- bert.pooler.dense.weight – 0,00004
- bert.classifier.bias – 0,00004
- bert.classifier.weight – 0,00004
- num_train_epochs – 4

B.1.2 Datová sada Endava

BERT (doladěn), CLS – XGBoost

Nebylo možné najít lepší hyperparametry než výchozí hodnoty.

BERT (doladěn), CLS – Random Forest

- max_depth – 90
- min_samples_leaf – 2
- min_samples_split – 2
- bootstrap – False
- max_features – auto
- n_estimators – 244

BERT (doladěn) – Klasifikační hlava

- layer_0-2 – 0,000022080483651216304
- layer_2-4 – 0,000090043896086547154
- layer_4-6 – 0,00000858214300390067
- layer_6-8 – 0,00014879468995776326
- layer_8-10 – 0,00011934285747053416
- layer_10-12 – 0,000037177589832433865
- bert.pooler.dense.bias – 0,0007768963552119172
- bert.pooler.dense.weight – 0,0008847370870839356
- bert.classifier.bias – 0,0008916967864083148
- bert.classifier.weight – 0,0002596315168962946
- num_train_epochs – 4

B.2 Doporučení řešitele pro tiket

BERT (doladěn), CLS – Logistic Regression

- C – 0.5714285714285714
- solver – saga

TF-IDF – XGBoost

- max_depth – 15
- learning_rate – 0,1
- subsample – 0,9
- colsample_bytree – 0,6

- colsample_bylevel – 0,6
- n_estimators – 300

BERT (doladěn) – Klasifikační hlava

Nebylo možné najít lepší hyperparametry než výchozí hodnoty.