VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VYUŽITÍ DOLOVÁNÍ DAT PRO IDENTIFIKACI PLATEB

USE OF DATA MINING FOR PAYMENT IDENTIFICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. STANISLAV BARTOŠ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Bartoš Stanislav, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Využití dolování dat pro identifikaci plateb**
Use of Data Mining for Payment Identification
Kategorie: Data mining

Zadání:

1. Seznamte se s metodami získávání znalostí z dat (data mining).
2. Seznamte se s dostupnými knihovnami na podporu dolování z dat v jazyce Python.
3. Zvolte vhodnou metodu pro identifikaci plateb, zejména při absenci některého z údajů. Získané informace poté budou využity pro spárování platby s neúplnými údaji s příslušnou fakturou.
4. Navržené řešení implementujte tak, aby demonstrovalo využití prostředků jazyka Python pro data mining.
5. Implementaci otestujte na datech firmy Platební instituce Roger, a.s. a proveďte experimenty vyhodnocující úspěšnost identifikace.
6. Zhodnoťte dosažené výsledky a další možnosti pokračování tohoto projektu.

Literatura:

- Han, J., Kamber, M.: Data Mining - Concepts and Techniques, 2nd Edition. Morgan Kaufmann Publishers, 2006.
- Layton, R.: Learning Data Mining with Python. Packt Publishing, 2015.

Při obhajobě semestrální části projektu je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 3. června 2020
Datum schválení: 24. října 2019

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací systému pro identifikaci plateb, a to i v případě, kdy chybí spolehlivý identifikátor jako například variabilní symbol. K řešení tohoto problému byly využity techniky z oblasti dolování dat, konkrétně klasifikace a predikce. Jedná se o firemní zadání diplomové práce pro firmu Platební instituce Roger a.s.

Abstract

This master thesis concentrates on design and implementation of a system for payment identification, even if the reliable identifier (e.g. variable symbol) is missing. Data mining techniques, such as classification and prediction, were used as a solution to this problem. This master thesis is company assignment for company "Platební instituce Roger a.s.".

Klíčová slova

identifikace plateb, dolování dat, Python, získávání znalostí, strojové učení, klasifikace, predikce, SVM, neuronová síť, rozhodovací strom

Keywords

payment identification, data mining, Python, knowledge discovery, machine learning, classification, prediction, SVM, neural network, decision tree

Citace

BARTOŠ, Stanislav. *Využití dolování dat pro identifikaci plateb*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Využití dolování dat pro identifikaci plateb

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Bartíka, Ph.D. Další informace mi poskytli zaměstnanci firmy Platební instituce Roger a.s. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Stanislav Bartoš
28. května 2020

Poděkování

Chtěl bych zde poděkovat zejména vedoucímu mé diplomové práce panu Ing. Vladimíru Bartíkovi, Ph.D., dále panu Ing. Tomáši Slobodníkovi za užitečné rady a konzultace a všem dalším zaměstnancům firmy Platební instituce Roger a.s., kteří mi pomohli se sběrem dat.

Obsah

Kapitola 1

Úvod

Cílem této práce je vytvořit nástroj, který bude schopen přiřadit platby k příslušným fakturám a to i za předpokladu, že některý údaj, například variabilní symbol, chybí. Platí, že jednou platbou může být uhrazeno více faktur, zároveň ale i jedna platba může znamenat úhradu pouze části faktury. To může nastat například v případě, kdy je faktura po splatnosti, ale plátce nedisponuje dostatečnými prostředky k úhradě celé faktury, a proto ji uhradí alespoň částečně.

Identifikace plateb jako taková není triviální, neboť neexistuje spolehlivý způsob, jak platbu automaticky identifikovat. V některých případech je možno použít pro identifikaci variabilní symbol, který ale často chybí – například u zahraničních plateb není vůbec. Datum splatnosti rovněž není jednoznačné, protože platba může dorazit dříve, ale i později. Stejně tak ani nominální hodnotu nelze k tomuto účelu použít, nemusí totiž sedět vzhledem k vazbě M:N mezi fakturami a platbami.

Vzhledem k tomu, že žádný údaj neidentifikuje platbu zcela stoprocentně, byl zvolen jako způsob řešení data mining (česky dolování dat). Tímto způsobem samozřejmě nepůjde identifikovat platbu zcela automaticky, nicméně se alespoň může podařit zmenšit množinu potenciálních faktur, ke kterým se úhrada vztahuje. Cílem mé diplomové práce je tedy prozkoumat metody, které data mining nabízí, a následně experimentálně zjistit, která metoda, případně kombinace metod, se jeví jako nejpřesnější. Vzhledem k tomu, že se jedná o neprozkoumané teritorium, samozřejmě může nastat i situace, že žádná metoda nebude dávat výsledky s rozumnou přesností.

V rámci diplomové práce bude nejprve popsán obecný úvod do problematiky získáváni znalostí z dat v kapitole 2 *Data Mining*. Kapitola 3 *Dolování dat v jazyce Python* pak pojednává o zvolených technologiích pro dolování dat k řešení daného úkolu. Kapitola 4 obsahuje jednak popis trénovacích dat, ale i návrh algoritmu pro implementaci prediktoru. Samotná implementace je pak popsána v kapitole 5. Implementovaný prediktor bylo následně nutné otestovat na testovacích datech a přesnost jednotlivých implementovaných metod vyhodnotit – o tom pojednává kapitola 6 *Experimenty a testování*. Poslední kapitolu pak představuje kapitola 7 *Závěr*, která obsahuje nejen zhodnocení dosažených výsledků, ale i nastínění možných pokračování tohoto projektu.

Kapitola 2

Data Mining

Tato kapitola obsahuje úvod do problematiky Data Miningu a zevrubný popis metod, které nabízí. Nejedná se ovšem o ucelený přehled všech oblastí dolování dat, nýbrž spíše o rozbor metod, které by s vysokou pravděpodobností mohly posloužit při řešení uvedeného problému. Informace obsažené v této kapitole vychází z [3], [1], [4] a [12].

2.1 Úvod do problematiky

Pojem *Data Mining* může být definován mnoha způsoby. Dle [3] je Data Mining proces odhalování zajímavých vzorů a znalostí z rozsáhlého množství dat. Zdrojem dat mohou být databáze, datové sklady, ale i web. Někdy se jako data mining označuje pouze část procesu získávání znalostí, celý proces se pak označuje jako *Knowledge discovery from data* (KDD). Pravděpodobně kvůli délce tohoto termínu se však pro označení celého procesu ujal spíše termín *Data Mining* a tak je tento pojem chápán i v této diplomové práci.

2.1.1 Proces dobývání znalostí

Proces získávání znalostí z dat sestává z několika částí, z nichž každá má svá specifika. Zde jsou dílčí kroky uvedeny stručně, podrobnější popis pak bude následovat v dalších podkapitolách.

1. **Čištění** - odstranění nekonzistentních a zašuměných dat
2. **Integrace** - kombinace dat z více zdrojů
3. **Výběr dat** - výběr relevantních dat pro analýzu
4. **Transformace** - transformace dat do vhodné podoby prostřednictvím agregačních či sumarizačních funkcí
5. **Dolování dat** - vlastní aplikace metod pro získávání znalostí
6. **Vyhodnocení vzorů** - identifikace zajímavých vzorců, které představují nově nabytou znalost
7. **Prezentace znalosti** - představení získaných znalostí uživateli pomocí vizualizačních technik

Kroky 1 až 4 se často označují termínem *předzpracování dat*, protože se jedná o přípravu dat před vlastním dolováním znalostí. Kroky čištění a integrace jsou pak často prováděny zároveň a jejich výstup je uložen v tzv. *datovém skladu*, což je z definice dle [3] datový repozitář, jenž integruje informace z vícero zdrojů pod jednotným schématem. Zároveň typicky obsahuje data za určité historické období; tato data jsou navíc nějakým způsobem agregována. Například místo ukládání všech detailů o každé transakci je uložen jen počet transakcí na položku a podobně.

2.1.2 Typické úlohy

Úloh, jež lze řešit pomocí data miningu, je relativně velké množství. Typy úloh lze obecně rozdělit na 2 podkategorie – *prediktivní* a *deskriptivní*. Deskriptivní úlohy zkoumají charakteristické vlastnosti dat v rámci datové množiny, oproti tomu prediktivní úlohy se snaží o předpovědi na základě analýzy současných, případně historických dat.

Následuje stručný přehled jednotlivých typů úloh, přičemž klasifikace bude podrobněji rozebrána dále, neboť bude využita při řešení daného problému.

Charakterizace dat

Pod tímto typem úlohy se skrývá sumarizace obecných vlastností cílové třídy. Data odpovídající této třídě jsou většinou získána pomocí dotazu. Například charakteristické vlastnosti produktů, jejichž prodeje stouply o 10% ve srovnání s předchozím rokem, lze získat SQL dotazem na databázi prodejů.

Diskriminace dat

Diskriminace dat je porovnání obecných vlastností cílové třídy oproti vlastnostem objektů jedné nebo více tříd kontrastních. Cílové a kontrastní třídy může specifikovat uživatel a k získání jejich odpovídajících objektů může být užito SQL dotazu. Například uživatele může zajímat, čím se od sebe odlišují produkty, jejichž prodeje stouply o 10% oproti produktům, jejichž prodeje poklesly o více než 30%.

Asociační pravidla

V těchto typech úloh jde o nalezení vzorů, jež se v rámci datové množiny vyskytují s relativně velkou četností. Existuje mnoho druhů frekventovaných vzorů. Frekventované množiny pak typicky odkazují na množinu položek, jež se v rámci datové množiny vyskytují často společně. Opakující se posloupnosti v datech – například skutečnost, že zákazník si poté, co si koupil digitální fotoaparát, koupí i paměťovou kartu – se označují jako frekventované sekvenční vzory. Dalším zástupcem jsou frekventované podstruktury. Podstrukturou rozumíme různé strukturované formy jako grafy, stromy a podobně. Pokud se tyto struktury v rámci datové množiny vyskytují často, pak jsou označovány jako frekventované strukturální vzory. Dolování frekventovaných vzorů umožňuje odhalení zajímavých skutečností o asociacích a korelacích mezi daty.

Výsledkem analýzy frekventovaných vzorů je typicky tzv. *asociační pravidlo* ve tvaru:

$$buys(X, "computer") \Rightarrow buys(X, "software")[support = 1\%, confidence = 50\%]$$

Podíl dané skutečnosti na celkovém počtu analyzovaných dat se označuje jako *support*, neboli podpora. *Confidence* (česky spolehlivost) pak značí v kolika procentech případů z těch,

kdy nastala situace na levé straně pravidla (zákazník si koupil počítač), zároveň nastala i situace popsaná na pravé straně pravidla (zákazník si koupil software). Toto konkrétní pravidlo lze tedy interpretovat tak, že pokud si zákazník X koupil počítač, pak si v polovině případů koupil i nějaký software, přičemž počítač společně se softwarem si koupil v 1% případu ze všech.

Jestliže se na obou stranách pravidla vyskytuje pouze jeden predikát (*buys*) hovoříme o jednodimenzionálním asociačním pravidlu, pakliže se predikátů v pravidle vyskytuje více jedná se o vícedimenzionální asociační pravidlo.

Asociační pravidla typicky nejsou brána v potaz, pokud hodnoty podpory nebo spolehlivosti nepřesáhnou stanovené minimální hranice.

Klasifikace a regrese

Klasifikace je proces hledání *modelu* nebo funkce, které co možná nejlépe rozlišují či popisují třídy dat. Model je odvozen na základě analýzy tzv. *trénovacích dat*, což jsou data, jejichž třída je dopředu známá. Takto získaný model pak může být použit k predikci třídy dat, u kterých dopředu známá není. Model je reprezentován mnoha způsoby, například klasifikačními pravidly, rozhodovacím stromem, matematickou rovnicí nebo neuronovou sítí. Rozhodovací strom je stromová struktura, kde každý uzel představuje dotaz na hodnotu atributu a větve možné výsledky. Listové uzly pak značí výslednou třídu. Výhoda rozhodovacích stromů spočívá hlavně v jejich názornosti a snadné interpretovatelnosti. Neuronová síť je kolekce jednotek (neuronů), které jsou mezi sebou propojeny, přičemž každé propojení má nastavenou vlastní váhu.

Existuje mnoho dalších klasifikátorů jako Naivní bayesovský klasifikátor, SVM (*Support Vector Machines*) a nebo metoda k-nejbližších sousedů (*k-nearest-neighbor classification*). Detailnější popis těchto metod bude následovat v dalších podkapitolách.

Zatímco klasifikace se zaměřuje na predikci nominálních, neboli kategorických atributů, jež jsou z principu neporovnatelné, regrese cílí na predikci atributů, jejichž hodnoty jsou seřaditelné (typicky číselné hodnoty). Termínem predikce se tedy rozumí jak predikce kategorických atributů (třídy), tak numerických. Nejčastěji používaná metoda pro predikci numerických hodnot se nazývá regresní analýza, která je založena na statistice.

Shluková analýza

Na rozdíl od klasifikace a regrese, které k učení využívají trénovací množinu, u níž je výsledná třída známá, se shluková analýza používá zejména tam, kde takovou trénovací množinu nelze získat. Pomocí shlukování lze třídy vytvořit na základě podobnosti určitých objektů mezi sebou. Základním principem je maximalizovat podobnost objektů uvnitř shluků (tzv. *intraclass similarity*), ale zároveň minimalizovat podobnost mezi objekty různých shluků (tzv. *interclass similarity*). Každý shluk je poté chápán jako třída objektů.

2.2 Předzpracování dat

Reálná data obvykle pochází z různých heterogenních zdrojů, jsou objemná co do počtu záznamů a často neúplná. Proto je potřeba je předzpracovat dřív, než se zahájí samotné dolování dat. Je třeba zjistit, jaké typy atributů data obsahují, zda se jedná o spojité či diskrétní, jaké je rozložení jejich hodnot, případně jestli neobsahují nějaké odchylky (*outliers*).

Problémy, které mohou u reálných nastávat jsou jmenovitě neúplnost, tedy chybějící hodnota některého atributu, případně v datech celý atribut. Dále nepřesnosti, či šum, což značí chybné hodnoty, nebo jejich vychýlení oproti očekávání. V neposlední řadě se může dojít k rozporu zejména u kategorických atributů, což se označuje jako nekonzistence v datech.

Technik předzpracování dat existuje několik. Jmenovitě se jedná zejména o čištění, integraci, transformaci a redukci dat. Pomocí čištění dat se lze vypořádat s nepřesnostmi a šumem. Cílem integrace je spojit data z více heterogenních zdrojů do jednoho unifikovaného (např. datový sklad). Transformace zahrnuje části jako normalizace, případně užití agregačních funkcí, odstranění šumu a podobně. A konečně redukce slouží ke zmenšení počtu rysů například pomocí eliminace, agregace, či shlukování. Tyto techniky se vzájemně nevylučují, naopak je lze libovolně kombinovat mezi sebou. Podrobnější popis těchto technik bude následovat dále.

2.2.1 Čištění dat

Pod pojmem čištění dat se rozumí zejména doplňování chybějících hodnot, odstraňování šumu a chyb, identifikace či odstranění odlehlých hodnot a v neposlední řadě také řešení nekonzistencí v datech.

Chybějící hodnoty

Chybějící hodnoty u některých atributů mohou vést ke zkresleným výsledkům, a proto je potřeba se s tím nějak vypořádat. Řešení tohoto problému exisujte několik:

1. **Ignorování uspořádané n-tice** – tedy odstranění celého záznamu, kde chybí některé hodnoty. Toto není příliš efektivní, neboť tím vynecháváme data, která mohla být nějakým způsobem užitečná. Proto se tento způsob používá takřka výhradně pouze v případech, kdy chybí hodnota u atributu pro klasifikaci (*class label*).

2. **Doplnění hodnot ručně** – tento způsob je velmi časově náročný a při velké množině dat de facto neproveditelný.

3. **Globální konstanta** – chybějící hodnoty jsou nahrazeny konstantou, což ale může vést k chybným interpretacím. Například v tom smyslu, že tato nová konstanta bude považována za samostatnou kategorii, která se může zdát zajímavou a tedy ovlivnit výsledek.

4. **Užití statistické míry centrální tendence** – mezi takovéto míry patří například aritmetický průměr, medián, modus atd. Při normálním rozložení lze použít aritmetický průměr, na vychýlená data je efektivnější užití mediánu.

5. **Statistická míra na vzorky shodné třídy** – obdobně jako u předchozího způsobu je použit aritmetický průměr či medián, ale rozdíl je v tom, že se neaplikuje na celý rozsah hodnot daného atributu, ale pouze na hodnoty náležící ke stejné třídě.

6. **Nahrazení nejpravděpodobnější hodnotou** – toho může být dosaženo například regresí, Bayesovskou klasifikací, či rozhodovacím stromem. Popis těchto metod bude následovat dále viz sekce 2.3. Obecně se jedná o nejpoužívanější metodu, neboť vykazuje relativně dobrou přesnost a navíc je vysoká pravděpodobnost, že vazby a závislosti mezi atributy zůstanou zachovány.

Šum v datech

Šum představuje náhodnou chybu či odchylku zkoumané hodnoty. Může vznikat například chybou hardwaru, ale i lidským faktorem. Proces opravování takovýchto chyb se nazývá *vyhlazování* (*binning*). Pro řešení tohoto problému existuje několik následujících technik:

- **Plnění** (binning) vyhlazuje hodnoty v seřazených datech na základě okolních hodnot. Jedná se tedy o lokální vyhlazování. Seřazené hodnoty jsou rozděleny mezi určitý počet tzv. *košů* (angl. *bin*). Následně jsou hodnoty v koších nahrazeny jednou hodnotou – průměrem, či mediánem jednotlivých košů. Další možností pak je nahrazení hodnot některou z hraničních hodnot koše, tedy maximem, či minimem. Každá hodnota v koši je pak nahrazena tou bližší z hraničních hodnot. V jednom koši se pak tedy vyskytuje nikoliv jedna hodnota, nýbrž dvě.

  Běžně jsou hodnoty mezi koše rozděleny rovnoměrně, tudíž počet hodnot v koši je přibližně stejný, ovšem rozsah hodnot v jednom koši může být relativně velký. Proto se někdy košům přiděluje pevný rozsah hodnot, což ovšem může zase vést k velkému rozdílu v počtu hodnot v jednotlivých koších.

- **Regrese** – vyhlazování dat lze rovněž realizovat pomocí regrese. Jedná se de facto o hledání funkční závislosti mezi 2 a více atributy. *Lineární regrese* je založena na nalezení nejlepší přímky mezi dvěma atributy tak, aby pomocí hodnot jednoho bylo možno odhadovat hodnoty druhého. Vícenásobná lineární regrese pak zahrnuje více než dva atributy.

- **Analýza odlehlých hodnot** – k analýze odlehlých hodnot lze použít například shlukování. Hodnoty, které nelze zařadit do žádného shluku pak lze označit jako odlehlé.

Tyto metody mohou být užity i k diskretizaci či redukci dat. Například technika plnění (binning) zmenšuje počet rozdílných hodnot v rámci atributu, čímž redukuje složitost dolovacích metod.

2.2.2 Integrace dat

Integrace dat je de facto spojování dat z více různých zdrojů. Dobře provedená integrace eliminuje redundance a nekonzistence ve výsledné datové množině, čímž může zvýšit přesnost a rychlost následných dolovacích procesů. Redundance je obvyklým problémem, jenž se vyskytuje při integraci dat. Atribut je považován za redundantní, pakliže může být odvozen od jednoho či více jiných atributů.

Některé redundantní atributy lze detekovat pomocí tzv. *korelační analýzy*. Korelační analýza umožňuje změřit míru závislosti mezi dvěma danými atributy podle jejich hodnot. Pro nominální atributy se používá χ^2 (**chí-kvadrát**) test. Pro atributy numerické lze pak užít **korelační koeficient** nebo **kovarianci**.

Nutno podotknout, že i když atributy mohou být statisticky korelované, nemusí to nutně znamenat kauzalitu, tedy že A nezapříčiňuje B a obráceně. Například při analýze demografické databáze můžeme zjistit, že počet nemocnic v regionu je statisticky korelovaný s počtem krádeží aut, což ale neznamená, že by mezi těmito dvěma atributy existovala kauzalita. Ve skutečnosti je zde závislost se třetím atributem, a to velikostí populace, kde už kauzalita samozřejmě existuje.

χ^2 korelační test

Korelační závislost mezi dvěma nominálními atributy A a B lze zjistit pomocí χ^2 testu. Předpokládejme, že A má s různých hodnot $A_1, A_2, ...A_s$, obdobně B má r různých hodnot $B_1, B_2...B_r$. N-tice vzniklé z těchto dvou atributů A a B pak lze znázornit pomocí tzv. *kontingenční tabulky*, kde s hodnot atributu A bude tvořit sloupce a r hodnot atributu B řádky. Konkrétní uspořádanou dvojici (tedy i pole v tabulce), kdy atribut A nabývá hodnoty a_i a atribut B hodnoty b_j, označíme jako (A_i, B_j). Hodnota χ^2, neboli Pearsonova χ^2 statistika, se pak vypočte podle rovnice 2.1[3]

$$\chi^2 = \sum_{i=1}^{s}\sum_{j=1}^{r}\frac{(o_{ij} - e_{ij})^2}{e_{ij}}, \tag{2.1}$$

kde o_{ij} je reálně pozorovaná četnost výskytu uspořádané dvojice (A_i, B_j) a e_{ij} pak četnost předpokládaná podle poměrného zastoupení hodnot viz rovnice 2.2

$$e_{ij} = \frac{count(A = a_i) \times count(B = b_j)}{n}, \tag{2.2}$$

kde n je celkový počet záznamů, $count(A = a_i)$ označuje počet záznamů, kde hodnota atributu A je rovna a_i, obdobně $count(B = b_j)$ značí počet záznamů, kde atribut B nabývá hodnoty b_j. Suma v rovnici 2.1 je spočítána pro všech $r \times s$ buněk v kontingenční tabulce.

χ^2 statistika ověřuje pravdivost předpokladu, že atributy A a B jsou vzájemně nezávislé. Tuto hypotézu testujeme podle tzv. kritické hodnoty v závislosti na hladině významnosti a stupni volnosti, který se spočte jako $(r-1)\times(s-1)$. Konkrétní kritické hodnoty pro testování hypotézy lze nalézt prakticky v kterékoliv učebnici statistiky. V případě, že je hodnota χ^2 větší než tato kritická hodnota, pak předpoklad nezávislosti můžeme zamítnout a prohlásit atributy za statisticky korelované.

Pearsonův korelační koeficient

Korelaci dvou numerických atributů A a B můžeme určit pomocí tzv. Pearsonova korelačního koeficientu. Ten se spočte podle vzorce 2.3

$$r_{A,B} = \frac{\sum_{i=1}^{n}(a_i - \bar{A})(b_i - \bar{B})}{n\sigma_A\sigma_B} = \frac{\sum_{i=1}^{n}(a_i b_i) - n\bar{A}\bar{B}}{n\sigma_A\sigma_B}, \tag{2.3}$$

kde n je počet n-tic, a_i a b_i představují hodnoty atributů A respektive B, $\bar{A}$ a $\bar{B}$ značí jejich střední hodnoty a σ_A, σ_B pak jejich směrodatné odchylky. $\Sigma(a_i b_i)$ je pak součet součinů hodnot atributů všech n-tic – tz. pro každou n-tici se hodnota atributu A vynásobí s hodnotou atributu B a všechny tyto součiny se sečtou. Platí, že $-1 \leq r_{A,B} \leq 1$.

Pakliže je $r_{A,B}$ větší než 0, jsou atributy pozitivně korelované, což znamená, že pokud poroste hodnota A, pak poroste i hodnota B a obráceně. Pakliže je $r_{A,B}$ větší než 0, jsou atributy negativně korelované, tedy s rostoucí hodnotou A bude hodnota B klesat. Platí, že čím je absolutní hodnota $r_{A,B}$ větší, tím je korelace silnější. Vysoká hodnota korelace může indikovat redundanci mezi atributy, a tedy že atribut A (nebo B) může být potenciálně odstraněn. Pokud je hodnota blízko nuly, pak lze říci, že atributy jsou na sobě nezávisle. Jak již bylo zmíněno, platí, že statistická korelace mezi dvěma atributy nemusí nutně znamenat jejich kauzalitu.

2.2.3 Transformace dat

Cílem transformace dat je převést data do podoby vhodné pro získávání znalostí. Existuje několik postupů a strategií, které se k tomuto účelu běžně využívají, následuje přehled těch nejznámějších:

- **Vyhlazování** – princip této fáze spočívá v odstranění šumu. Běžně aplikované techniky již byly zmíněny při popisu fáze čištění – zahrnují shlukování, regresi, či metodu tzv. plnění košů (angl. *binning*).
- **Konstrukce rysů** – princip této techniky je relativně zřejmý již z názvu, spočívá v tvorbě nových rysů, jež jsou vypočteny na základě rysů stávajících.
- **Agregace** – tato strategie zahrnuje aplikaci různých sumarizačních a agregačních funkcí, což zajišťuje vyšší úroveň abstrakce.
- **Normalizace** – cílem této metody je rozmístit data do určitého intervalu tak, aby například velmi vysoké hodnoty neměly při klasifikaci vyšší vliv. Mezi nejznámější typy normalizace patří zejména dva typy. Prvním je *min-max* normalizace, která mapuje hodnoty do požadovaného intervalu přímo na základě minimální a maximalní hodnoty v datech. Druhým typem je pak *z-skóre* normalizace, která zajistí transformaci do požadovaného intervalu nepřímo na základě průměru a směrodatné odchylky – tato metoda se vyplatí zejména v případě, že minimum či maximum, které se může v datech objevit, není předem známé, což je obvyklá situace.
- **Diskretizace** – cílem je převod spojitého číselného datového typu na kategorický typ. Techniky zahrnují tzv. plnění košů (angl. *binning*), shlukování, diskretizaci na základě analýzy histogramu a podobně.

2.3 Klasifikace a Predikce

V této podkapitole budou podrobněji rozebrány jednotlivé metody, které se používají pro klasifikaci a predikci. Informace zde obsažené vychází z [3], [1], [4] a [12].

Cílem klasifikace je extrakce modelů popisujících důležité třídy v rámci dat. Tyto modely zvané klasifikátory je pak možno využít pro predikci kategorických atributů jakožto tříd. Pro predikci numerických atributů existuje regresní analýza, ale zde bude popsána výhradně klasifikací, neboť ta bude využita při řešení daného problému.

Proces klasifikace je možno rozdělit na dvě fáze – učení (též trénování) a testování. Ve fázi učení je sestaven klasifikátor na základě tzv. trénovací množiny, což je množina dat, na které učení probíhá. Na základě trénovací množiny, kterou máme k dispozici můžeme rozlišit dva různé principy. Pokud je u dat v trénovací množině jejich příslušnost ke třídám dopředu známá, hovoříme o učení s učitelem (*supervised learning*), v případě, že dopředu známá není, jedná se o učení bez učitele (*unsupervised learning*), tedy vlastně shlukování. V případě učení bez učitele nemusí být dopředu známý ani počet tříd.

Na první fázi klasifikace, tedy učení, je možno nahlížet jako na proces hledání funkce $y = f(\mathbf{X})$, která dokáže predikovat výslednou třídu y pro dané $\mathbf{X}$. Touto funkcí mohou být matematické vzorce, ale i klasifikační pravidla, rozhodovací stromy a podobně.

Ve druhé fázi je získaný model použit pro klasifikaci. V tomto kroku se vyhodnocuje přesnost klasifikátoru. Pokud by se k měření přesnosti použila stejná data jako pro testování, pak by výsledná přesnost byla s velkou pravděpodobností lepší než realita, neboť při učení

může klasifikátor využít například některých anomálii v trénovací množině, které ale v celé množině dat nalézt nelze. Proto se pro měření přesnosti volí odlišná množina dat, tzv. testovací množina. Typicky jsou tedy data, která máme k dispozici, rozdělena v určitém poměru na dvě části – na trénovací a testovací množinu.

Důležitým pojmem je zmíněná přesnost klasifikátoru. Je to de facto procentuální míra úspěšnosti klasifikátoru, tedy míra správně klasifikovaných dat. Pro všechna data je předem známá třída porovnána se třídou, kterou naučený klasifikátor predikuje. Pokud je přesnost klasifikátoru prohlášena za uspokojivou, může být tento naučený klasifikátor využit pro predikci třídy u dat, jejichž třída je dopředu neznámá.

2.3.1 Rozhodovací stromy

Rozhodovací strom je stromová struktura, kde každý nelistový uzel představuje porovnání hodnoty atributu, větve pak představují výsledky tohoto porovnání a z listových uzlů nakonec získáme výslednou třídu.

Klasifikace n-tice, u níž třídu neznáme, probíhá tak, že hodnoty jejích atributů jsou testovány naučeným rozhodovacím stromem, přičemž je zachycena cesta od kořenového uzlu k listovému, který obsahuje výslednou třídu pro danou n-tici. Rozhodovací stromy lze snadno převést na klasifikační pravidla.

Vytváření rozhodovacího stromu

Princip rozhodovacího stromu spočívá ve výběru atributů, které data nejlépe rozřazují do tříd – na kořenové úrovni bude porovnání atributu, který rozřazuje data nejlépe, a tak dále pro každý podstrom. Tímto způsobem je snížen počet porovnání, která je třeba při klasifikaci provést. Celý algoritmus je zjednodušeně popsán pseudokódem 1.

Důležitým pojmem je dělící kritérium. Toto kritérium nám říká, který atribut vybrat pro přiřazení k uzlu N a kromě toho udává i tzv. dělící bod (*split-point*), či dělící podmnožinu (*split-subset*). Dělícím bodem může být například konkrétní číslo u numerických atributů, dělící podmnožinou pak několik vybraných hodnot pro kategorický atribut. V případě, že je povoleno více větví stromu nežli dvě, tedy není strom omezen na binární, pak je u kategorických atributů možno provést rozvětvení odpovídající možným hodnotám atributu, tedy jedna větev pro jednu hodnotu.

Výběr atributu s nejvyšší rozlišovací schopností

Klíčovým aspektem je funkce pro výběr atributů *Select*. Těch existuje několik – například metoda *information gain*, kterou používá algoritmus klasifikace rozhodovacím stromem ID3. Tato metoda volí atribut, který dosáhne nejlepšího hodnocení zisku informace, což je hodnota získaná jako rozdíl mezi očekávanou informací potřebnou ke klasifikaci (závisí na poměrném zastoupení tříd v rámci D) a informací kterou ve skutečnosti potřebujeme, aby byla všechna data klasifikována správně.

Další funkcí, kterou používá například algoritmus CART je tzv. *Gini index*. Hlavní myšlenkou této metody je de facto redukce míry špatně klasifikovaných n-tic při náhodném výběru třídy (tzv. *impurity*) viz rovnice 2.4 .

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2 \tag{2.4}$$

Algoritmus 1: Algoritmus generování rozhodovacího stromu [3]

Vstupy : D – množina trénovacích dat
A – seznam kandidátních atributů
Select – metoda pro výběr nejlepšího atributu a dělícího kritéria
Výstupy: Rozhodovací strom
Function *Generate_decision_tree(D,A)*:
  vytvoř uzel N;
  if *všechna data v D patří do téže třídy C* **then**
    return *N jako listový uzel třídy C;*
  end
  if *seznam A je prázdný* **then**
    return *N jako listový uzel nejvíce zastoupené třídy v D;*
  end
  vyber nejlepší dělící kritérium pomocí zvolené metody $Select(D, A)$;
  přiřaď uzlu D zvolené kritérium;
  if *Vybraný atribut je diskrétní* **and not** *pouze binární strom* **then**
    $A = A-\{$zvolený atribut$\}$;
  end
  foreach *výsledek j dělícího kritéria* **do**
  // dělení D a tvorba podstromů pro každou část
    D_j = množina dat splňujících výsledek j;
    if *seznam* D_j *je prázdný* **then**
      přidej k N listový uzel nejvíce zastoupené třídy v D;
    else
      přidej k N uzel vrácený rekurzivním voláním
       $Generate\_decision\_tree(D_j, A)$;
    end
  end
  return *N;*

Cílem je minimalizovat tuto hodnotu, což nastane v momentě, kdy většina n-tic v jedné množině patří ke stejné třídě, tedy je klasifikována správně. Strom této metody je striktně binární – u kategorických atributů je potřeba použít dělící podmnožinu *split-subset*. Při výběru atributů je brán v potaz Gini index nového rozdělení na základě poměru velikostí vzniklých podmnožin, viz vzorec 2.5

$$Gini_A(D) = \frac{|D1|}{|D|}Gini(D_1) + \frac{|D2|}{|D|}Gini(D_2) \tag{2.5}$$

Vybrán je atribut, který má tuto hodnotu nejmenší, respektive největší redukci Gini indexu viz rovince 2.6

$$\Delta Gini(A) = Gini(D) - Gini_A(D) \tag{2.6}$$

Náhodný les

Jako náhodný les se označuje klasifikátor, který je složen z více rozhodovacích stromů. Každý z dílčích rozhodovacích stromů je vytvářen na základě podmnožiny trénovací množiny, přičemž prvky této podmnožiny jsou z trénovací množiny vybrány náhodně. Při samotném vytváření stromu pak existují dva přístupy – buď je dělící atribut vybrán z náhodně zvolené podmnožiny existujících atributů přímo, nebo nepřímo tak, že jsou vytvořeny nové atributy pomocí lineární kombinace náhodně vybraných atributů stávajících. Klasifikace pak probíhá tak, že každý z dílčích stromů hlasuje o výsledné cílové třídě, načež je vybrána třída s nejvíce hlasy (modus).

Výhodou klasifikátoru typu náhodný les je zejména odolnost vůči výskytu chyb či odlehlých dat v trénovací množině, a navíc i odolnost proti přetrénování.

2.3.2 Bayesovská klasifikace

Bayesovská klasifikace je založena na statistice, přičemž predikce je prováděna na základě pravděpodobnosti, že daná n-tice náleží ke konkrétní třídě. Bayesova klasifikace je postavena na Bayesově větě o podmíněné pravděpodobnosti, jež je formulována rovnicí 2.7

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \tag{2.7}$$

Výsledkem této rovnice $P(H|X)$ je podmíněná pravděpodobnost jevu H (hypotézy) v případě, že nastal jev X (pozorovaný jev, který je nám známý). Obdobně $P(X|H)$ je podmíněna pravděpodobnost jevu X, pokud nastal jev H. $P(X)$ a $P(H)$ jsou pak nepodmíněné pravděpodobnosti příslušných jevů.

Naivní bayesovský klasifikátor

Naivní bayesovský klasifikátor je velmi běžný klasifikátor. Jeho oblíbenost pramení zejména z jeho relativně vysoké efektivity při klasifikaci rozsáhlých dat, ve smyslu velkého počtu atributů (rysů). Hlavní myšlenkou tohoto klasifikátoru, kvůli níž také dostal přízvisko naivní, je zjednodušující předpoklad, že každý atribut je nezávislý na ostatních. Díky tomuto předpokladu je tak efektivní na datech s velkým počtem atributů, na druhou stranu je potřeba věnovat eliminaci závislých atributů patřičnou pozornost při předzpracování dat (např. pomocí zmíněné statistické korelace). Vzhledem k tomu, že atributy jsou nezávislé, pak můžeme spočítat pravděpodobnost $P(X|C_k)$, tedy pravděpodobnost konkrétního vektoru rysu X pro třídu C_k, pouhým vynásobením pravděpodobností jejich konkrétních hodnot mezi sebou.

Hledáme tedy třídu C_k s nejvyšší pravděpodobností podmíněnou konkrétním vektorem rysů (atributů) X, tedy $P(C_k|X)$. Tu spočítáme podle Bayesovy věty 2.7, přičemž jmenovatel $P(X)$, můžeme vynechat, neboť je pro všechny třídy stejný. Výběr konkrétní třídy je tedy proveden na základě vzorce 2.8

$$Y_{pred} = \underset{k\in\{1,2,..,m\}}{\operatorname{argmax}} P(C_k) \prod_{i=1}^{n} P(X_i|C_k) \tag{2.8}$$

kde Y_{pred} je predikovaná třída, X vektor rysů, C množina tříd, n počet rysů a m představuje počet tříd. Je tedy vybrána třída C_k s nejvyšší hodnotou pravděpodobnosti $P(C_k|X)$. Podrobnější odvození tohoto vzorce je možno nalézt v [1]. Pravděpodobnost $P(C_k)$ se spočítá podle poměrného zastoupení tříd v trénovací množině, $P(X_i|C_k)$ se počítá odlišně

v závislosti na tom, zda je onen konkrétní atribut spojitý, či diskrétní – u spojitých je pravděpodobnost dána na základě Gaussova normálního rozložení, u diskrétních (kategorických) pak rozdělením multinomickým na základě poměrného zastoupení hodnoty daného atributu v rámci konkrétní třídy.

V případě, že by nás zajímala i konkrétní hodnota pravděpodobnosti predikované třídy, lze ji velmi snadno dopočítat – před výběrem maxima (argmax) je potřeba ještě provést dělení jmenovatelem z Bayesovy věty 2.7, tedy nepodmíněnou pravděpodobností vektoru rysů $P(X)$.

2.3.3 Algoritmus k-nejbližších sousedů

Algoritmus k-nejbližších sousedů (*k-NN*) patří mezi nejjednodušší klasifikační metody, neboť zkoumá pouze K nejbližších n-tic. Výsledná třída je pak určena hlasováním (*majority voting*) v rámci množiny K sousedů. Algoritmus potřebuje dva povinné parametry – velikost kardinality K a míru podobnosti respektive vzdálenosti mezi prvky – často používanou mírou je například Euklidovská vzdálenost.

2.3.4 Neuronové sítě

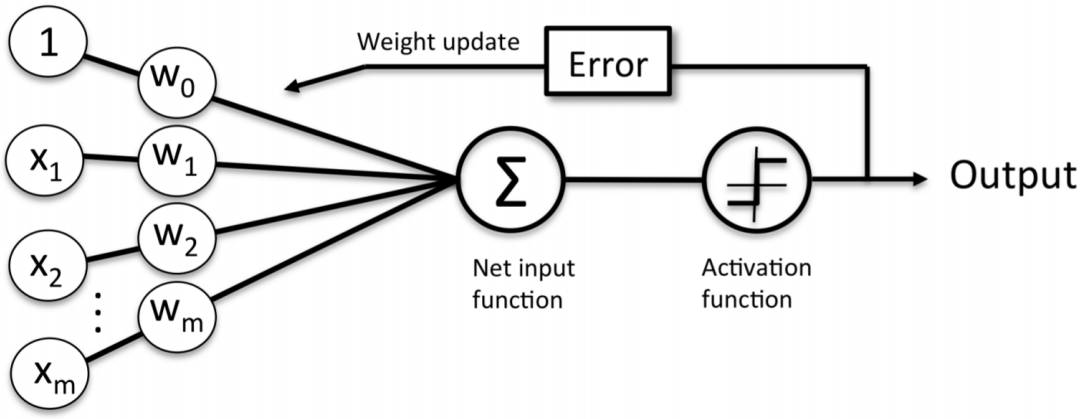
Neuronová síť patří k algoritmům inspirovaným přírodou – zde konkrétně biologickým neuronem. Zjednodušeně řečeno je neuron jednotka, která má mnoho vstupů, ale pouze jeden výstup. Každému ze vstupů je přiřazena váha. Neuronová síť je potom obecně propojení neuronů, kde výstupní hodnota neuronu může být vstupem dalších neuronů, přičemž každý má svou váhu. Učení neuronové sítě probíhá hledáním optimálního nastavení vah. To může trvat poměrně dlouhou dobu, což představuje jednu z největších nevýhod neuronových sítí. Velmi používanou metodou pro klasifikaci je tzv. hluboké učení (*deep learning*), což jsou obvykle dopředné vícevrstvé neuronové sítě. Dopředné znamená, že v propojení neuronů neexistuje cyklus, vícevrstvé pak, že v rámci sítě lze vypozorovat více vrstev tak, že výstup jedné vrstvy je vstupem další vrstvy. U hlubokých sítí je typicky kromě výstupní vrstvy ještě i jedna či více vrstev skrytých. Jedním z nejvíce používaných algoritmů pro trénovaní neuronové sítě je *backpropagation*.

Perceptron

Nejjednodušším typem neuronu je *perceptron*. Jeho nevýhodou však je neschopnost klasifikovat nelineárně oddělitelná data, na druhou stranu tento problém lze vyřešit sítí s více perceptrony. Schéma perceptronu je znázorněno na obrázku 2.1. Výpočet výstupu perceptronu probíhá tak, že nejprve je spočítán potenciál pomocí bázové funkce – u perceptronu Σ. Následně je spočítán výstup pomocí aktivační funkce – u perceptronu skoková funkce (*Heavisideova funkce*) podle určité hodnoty tzv. *threshold* Θ. Pro zjednodušení lze nastavit bias $x_0 = 1$ a $w_0 = \Theta$, takže hodnota výstupu se pak určí porovnáním s 0, viz rovnice 2.10.

$$z = w_0 x_0 + w_1 x_1 + ... w_m x_m = \mathbf{w}^T \mathbf{x} \tag{2.9}$$

$$\Phi(z) = \begin{cases} 1 \text{ pokud } z \leq 0 \\ -1 \text{ jinak} \end{cases} \tag{2.10}$$



Obrázek 2.1: Schéma perceptronu [12]

Algoritmus Backpropagation

Backpropagation je velmi efektivní a často používaný algoritmus pro trénování vícevrstvých dopředných neuronových sítí. Princip algoritmu je následující – nejprve je spočítán výstup neuronové sítě, který je následně porovnán s očekávanou hodnotou výstupu, na základě čehož je spočítána chyba. Ta je následně šířena od výstupní vrstvy přes všechny skryté vrstvy zpět, přičemž jsou modifikovány váhy tak, aby se výsledná chyba zmenšila. Jedná se tedy o hledání lokálního minima chyby, což vede k možnosti uváznutí v lokálním minimu bez nalezení minima globálního.

2.3.5 Support vector machines

Klasifikační metoda *Support vector machines* (dále SVM) je de facto obdoba perceptronu. Základní odlišnost spočívá v tom, že cílem trénování perceptronu je minimalizace chyby klasifikace, naproti tomu cílem metody SVM je nalézt tzv. maximální odstup (anglicky *margin*) mezi dvěma třídami. Princip tedy spočívá v nalezení optimální *dělící nadroviny*, tak, aby odstup mezi třídami byl co největší. Odstup je pak definován jako vzdálenost mezi dělící nadrovinou a nejbližšími prvky trénovací množiny – tyto prvky se nazývají *podpůrné vektory* (anglicky *support vectors*), od nichž je odvozen název této metody.

Tento princip je však aplikovatelný pouze na lineárně oddělitelná data. Pro lineárně neoddělitelná data je třeba provést tzv. *kernelovou* transformaci, tedy nelineární transformaci trénovacích dat do prostoru s větší dimenzí tak, aby bylo možné data oddělit lineárně. Zjednodušeně řečeno to znamená vytváření nových pseudo-rysů, dokud data nebudou lineárně oddělitelná. Je dokázáno, že tato transformace vždy existuje. Kernel je pak definován jako skalární součin dvou tzv. kernelových funkcí – nejčastěji používané kernelové funkce jsou například lineární, nebo radiální bázová funkce (rbf). Následně je provedena výše zmíněná optimalizační úloha hledání optimální dělící nadroviny.

Tato metoda je principiálně použitelná pro klasifikaci do dvou tříd, neboť dělící nadrovina rozdělí daný hyperprostor pouze na dvě části, nicméně existují i rozšíření pro klasifikaci do více tříd.

2.4 Vyhodnocení přesnosti modelů

Po natrénování klasifikátorů je nutné vyhodnotit přesnost, s jakou jsou schopné klasifikovat jednotlivé testovací vzorky. Trénovací a testovací množina by měly mít prázdný průnik, aby bylo možné ověřit, jak se klasifikátor chová na neznámé množině dat. Pro měření schopnosti klasifikátoru správně klasifikovat existuje mnoho různých metrik, zde budou uvedeny ty nejznámější. Na základě zvolených metrik je pak možno vybrat model, který dosahuje nejlepších výsledků, což se označuje termínem *model selection*.

2.4.1 Základní pojmy

Pro pochopení principu metrik pro vyhodnocení přesnosti modelů bude nejprve nutné si zadefinovat některé pojmy s tím související. Jedná se o čtyři pojmy související s výsledky predikce, které představují základní stavební kameny mnoha metrik.

Tyto pojmy pracují s tzv. pozitivní a negativní třídou. Nutno podotknout, že ač jsou tyto pojmy a metriky obvykle spojovány s binární klasifikací, jsou využitelné i pro klasifikaci do více tříd – jako pozitivní se označí jedna zájmová třída, zbylé se označí jako negativní a následně jsou spočítány metriky pro aktuálně vybranou třídu. Poté se jako zájmová pozitivní třída označí další ze tříd a tak dále pro všechny zbývající třídy zvlášť. Výsledky metrik jsou pak vypočítány jako průměrné hodnoty (ať už nevážené, či vážené dle zastoupení tříd) z dílčích výsledků metrik pro jednotlivé třídy.

- *True positives* (TP) – tento termín představuje vzorky, jež byly správně klasifikovány jako pozitivní. TP je počet těchto vzorků.
- *True negatives* (TN) – vzorky, jež byly správně klasifikovány jako negativní, TN představuje jejich počet.
- *False positives* (FP) – vzorky, jež byly chybně klasifikovány jako pozitivní, FP značí počet těchto vozrků.
- *False negatives* (FN) – vzorky, jež byly chybně klasifikovány jako negativní, FN je jejich počet.

Dalším důležitým pojmem je tzv. *matice záměn* (anglicky *confusion matrix*). Ta shrnuje výše uvedené hodnoty do tabulky. Obecný formát matice záměn je znázorněn tabulkou 2.1. Matice záměn velmi názorně ukazuje výsledky klasifikace, nicméně se technicky vzato nejdená o metriku, neboť metrika je obecně funkce, jejíž výsledkem je číslo. Matici záměn je možno využít jak u binární klasifikace, tak u klasifikace do více tříd.

		Predikce		
		0	1	celkem
Skutečnost	0	TN	FP	P
	1	FN	TP	N
	celkem	P’	N’	P+N

Tabulka 2.1: Obecný formát matice záměn pro binární klasifikaci [3]

2.4.2 Metriky

Nyní následuje přehled některých nejznámějších metrik. Nejedná se ovšem o zevrubný rozbor všech metrik, záměrem je spíše seznámit se s některými nejpoužívanějšími z hlediska principu, který se obecně v metrikách pro vyhodnocení přesnosti modelů uplatňuje.

Metriky je obecně možné rozdělit do dvou skupin. První z nich představují ty, které pracují s finálními výsledky klasifikace, tedy zejména s výše uvedenými hodnotami TP, TN, FP, FN – tyto metriky se označují termínem *thresholded metrics*. Do této skupiny patří většina základních metrik – *accuracy*, F_1 *score*, *precision*, *recall*, atd. Druhou skupinu tvoří metriky, které nepracují přímo s výslednými predikovanými třídami, ale se skóre, kterého jednotlivé vzorky dosáhly, což je de facto pravděpodobnost příslušnosti k určité (většinou pozitivní) třídě. Tato skupina se označuje termínem *ranked metrics* a patří sem například metriky *AUC-ROC* či *AUC-PR*.[5]

Metrik pro vyhodnocování a porovnávání výsledků různých klasifikátorů existuje samozřejmě velké množství, zde následuje přehled jen některých nejznámějších a nejpoužívanějších:

- *accuracy* – tato metrika je asi nejznámější a nejpoužívanější. Jedná se de facto o míru správně klasifikovaných vzorků viz vzorec:

$$accuracy = \frac{TP + TN}{P + N}$$

- *error rate* – představuje míru chybovosti, tedy míru chybně klasifikovaných vzorků vzhledem k celkovému počtu:

$$error\ rate = \frac{FP + FN}{P + N}$$

- *precision* – míra vzorků, jež byly predikovány jako pozitivní a opravdu jsou pozitivní:

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

- *recall* – míra pozitivních vzorků, které zároveň byly predikovány jako pozitivní ze všech skutečně pozitivních:

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

- F_1 *score* – jedná se o harmonický průměr výsledků metrik *precision* a *recall*:

$$F_1\ score = \frac{2 \times precision \times recall}{precision + recall}$$

- *AUC-ROC* – celým názvem *area under curve ROC*. Křivka ROC (*Receiver operating characteristic*) je funkce závislosti míry *true positive rate* (TPR) na míře *false positive rate*. Míra TPR je stejná jako míra *recall*, $FP = \frac{FP}{N}$. AUC-ROC je pak obsah pod touto křivkou. Metrika AUC-ROC patří do skupiny *ranked metrics*, tedy metrik postavených na pravděpodobnosti příslušnosti k pozitivní třídě – též se označuje termínem skóre. Z hlediska významu se dá tato metrika interpretovat jako pravděpodobnost, že náhodný pozitivní vzorek bude mít vyšší skóre, než náhodně vybraný negativní vzorek.

Kapitola 3

Dolování dat v jazyce Python

V této kapitole budou rozebrány nástroje pro podporu dolování dat v jazyce Python. Informace v této kapitole vychází z [8], [1], [4] a [12].

Python je objektově orientovaný, multiplatformní jazyk. V porovnání s ostatními jazyky se vyznačuje stručností, díky čemuž je možný velmi rychlý vývoj. Python je v současnosti velmi populární jazyk pro dolování dat – spolu s jazykem R představují majoritu. Toto je dáno zejména velkou flexibilitou jazyka Python – neboť je velmi dobře integrovatelný s dalšími jazyky a zároveň existuje velké množství balíčků a rozšíření, pomocí kterých lze řešit širokou škálu problémů, včetně dolování dat. V této kapitole budou stručně popsány ty nejklíčovější.

3.1 NumPy

NumPy je základní knihovna pro analýzu dat v Pythonu. Poskytuje nový datový typ – homogenní multidimenzionální pole `ndarray`, včetně pestré škály velmi užitečných funkcí nad ním. To je velmi užitečné pro řešení problémů s maticemi a vektory.

Mezi velkou výhodu této knihovny patří dále zejména vektorizované funkce a operace, a to jak základní (například suma prvků pole) tak pokročilé – například operace lineární algebry, nebo statistické operace (například Pearsonův korelační koeficient). Vektorizované operace jsou implementovány v jazyku C a navíc je možno je vykonávat paralelně na prvcích pole – to ve svém důsledku přináší mnohdy i několika řádové zrychlení oproti situaci, kdy by tyto operace byly implementovány pomocí cyklu v jazyce Python. Nad polem `ndarray` pak pracují prakticky všechny knihovny pro data mining v Pythonu, je to základní ty pro trénovací množinu u prakticky všech klasifikačních metod.[9]

3.2 SciPy

Knihovna SciPy je nadstavba nad knihovnou NumPy – poskytuje řadu pokročilých vědeckých algoritmů nad vektory a maticemi, například z oblasti lineární algebry, algoritmy pro řídké matice, zpracování signálů a obrazu, optimalizace, Fourierovu transformaci, atd.

Kromě toho obsahuje SciPy kompletní implementace některých algoritmů pro shlukování – například k-means.[14]

3.3 Pandas

Knihovna Pandas nabízí především specifické datové struktury – `DataFrames` a `Series`. Struktura `DataFrames` umožňuje práci s tabulkami se sloupci různých datových typů – jedná se tedy o heterogenní dvoudimenzionální pole. To je zásadní rozdíl oproti homogennímu poli `ndarray` z knihovny NumPy. Homogenní struktura `Series` pak představuje de facto sloupec v oné tabulce. To umožňuje jednoduché nahrávání dat z více zdrojů, včetně předzpracování, tedy doplnění chybějících hodnot, transformace, agregace, redukce, atd.

Další velkou výhodou jsou – stejně jako u knihovny NumPy – vektorizované operace, což velmi urychlí proces předzpracování dat. Navíc je zde možnost implementovat vlastní funkce, které je následně možné aplikovat pro všechny prvky, případně podél některé z os například pomocí metody `apply`, kterou nabízí jak třída `DataFrame`, tak `Series`. Zrychlení je zde sice nižší než v případě vektorizovaných operací, nicméně pořád se jedná o citelné zrychlení oproti implementaci dané funkcionality pomocí cyklu v Pythonu.

Pro shrnutí tedy platí, že pokud je to možné, je nejlepší využít vektorizované operace knihovny NumPy, pokud ne, pak metody které nabízí knihovna Pandas pro vykonání na dané struktuře pro jednotlivé prvky (např. zmíněná metoda `apply`). Použití cyklu v jazyku Python pro tyto účely je radno se vyhnout, neboť není příliš efektivní, a pro velké množiny dat může jeho použití zásadním způsobem zpomalovat běh programu.[10]

3.4 Scikit-learn

Knihovna Scikit-learn představuje jádro řešení dolování dat. Nabízí vše od předzpracování dat, metody strojového učení – jak učení s učitelem, tak i učení bez učitele, validace, metriky přesnosti, atd. Použití této knihovny je velmi jednoduché, až triviální, viz příklad učení perceptronu 3.1. Knihovna scikit-learn dále plně podporuje práci jak s polem `ndarray` z knihovny NumPy, tak potažmo i se strukturou `DataFrame` z knihovny Pandas.

Knihovna Scikit-learn poskytuje prakticky všechny základní metody pro klasifikační úlohy, což její využití činí prakticky nezbytným. Trénování klasifikátorů je realizováno pomocí metody `fit`, predikce pak metodou `predict`. Kromě toho knihovna poskytuje funkci `cross_validate`, jež implementuje *křížovou validaci*, která se hodí zejména pro experimentování s různými klasifikátory a porovnávání jejich přesností. K tomuto účelu je rovněž velmi užitečná funkce na vykreslení matice záměn – `plot_confusion_matrix`.

Další užitečnou funkcionalitu představuje kódování kategorických atributů na datový typ float, což lze realizovat zejména pomocí tříd `OneHotEncoder`, či `OrdinalEncoder`. Velmi důležitou částí předzpracování, jejíž implementaci tato knihovna rovněž poskytuje, je normalizace, bez které spousta klasifikačních metod nefunguje správně, neboť mnoho metod vyžaduje normální Gaussovo rozložení s nulovým průměrem a jednotkovým rozptylem – tuto funkcionalitu poskytuje například třída `StandardScaler`. Tyto třídy lze stejně jako natrénovaný model pro predikci ukládat pomocí klasických knihoven pro serializaci objektů do binární reprezentace, například pomocí knihovny pickle, což je standardní knihovna jazyka Python.[11, 13]

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Perceptron

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

ppn = Perceptron(n_iter=40, eta0=0.1, random_state=0)
ppn.fit(X_train_std, y_train) # learning
y_pred = ppn.predict(X_test_std)
print('Misclassified samples: %d' % (y_test != y_pred).sum())
```

Výpis 3.1: Učení perceptronu pomocí knihovny Scikit-learn [12]

3.5 Keras a Tensorflow

Tensorflow je robustní knihovna pro strojové učení, zejména neuronové sítě, a podporuje paralelní běh jak na CPU, tak GPU. Nad touto knihovnou je pak postavena knihovna Keras, která poskytuje vysokoúrovňové API, pro práci s neuronovými sítěmi, díky čemuž urychluje vývoj aplikací využívající strojové učení, což se hodí zejména pro experimenty.

Navíc knihovna Keras poskytuje API pro podporu knihovny scikit-learn, díky čemuž je možné mít kompletní předzpracování realizované pomocí knihovny scikit-learn, a jako klasifikátor pak následně použít model neuronové sítě vytvořený pomocí knihovny Keras. Ukládání natrénovaného modelu je však odlišné od knihovny scikit-learn, neboť zde se použití knihovny nedoporučuje, místo toho se doporučuje spíše použít serializaci ve formátu HDF5.

Knihovna Keras umožňuje definici vlastní neuronové sítě, včetně jednotlivých vrstev – k dispozici jsou pro ilustraci například vrstvy dopředné, rekurentní, ale i konvoluční. Pro trénování modelu pro klasifikaci do dvou tříd, což je předmětem mé diplomové práce, se hodí zejména vrstvy dopředné, konvoluční sítě se používají zejména v oblasti zpracování signálů potažmo obrazu, rekurentní pak například pro predikci posloupnosti.[6, 16]

Kapitola 4

Návrh

V této kapitole bude navržen postup řešení daného problému. Je zapotřebí vymyslet způsob, na základě kterého by bylo možné určit, ke které faktuře daná transakce náleží. Jelikož mohou být obecně zdroje faktur i transakcí různé, bude nutné použít jednotný postup nezávisle na zdrojích. Systém by tak měl být modulární. Natrénovaný klasifikátor by měl být využitelný i pro faktury a transakce dalších firem, což je nutno taky zohlednit – například zvolením vhodné vnitřní reprezentace dat, k čemuž by mohla být využita datová struktura `DataFrames` z knihovny Pandas.

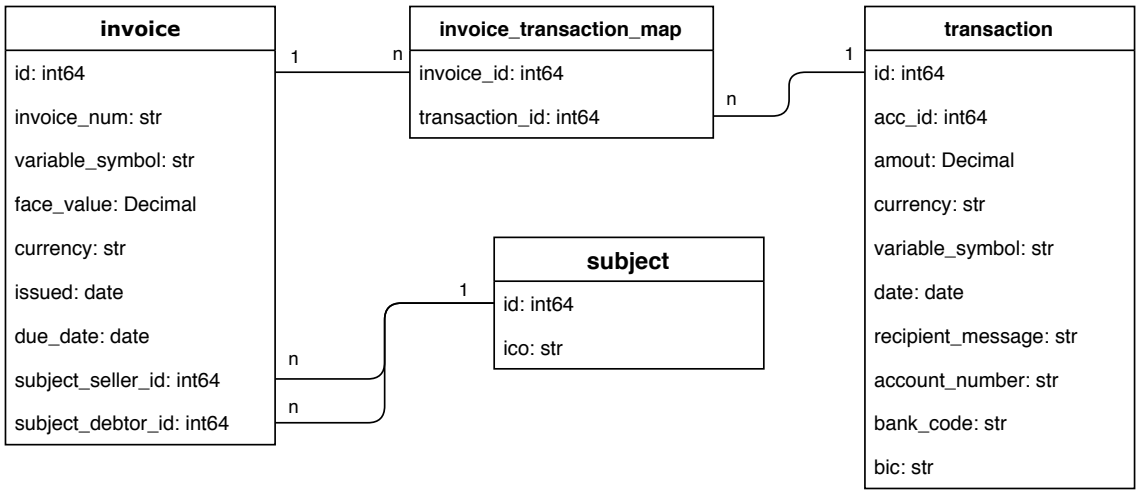
4.1 Popis dat

V současné době probíhá párování transakcí a faktur ručně. Faktury i transakce jsou uloženy v tabulce v databázi. Ukázalo se však, že chybí vazba mezi těmito tabulkami – to bylo nutné vyřešit hned v první fázi. Jelikož je vazba mezi těmito entitami M:N, byla vytvořena vazební tabulka. Výsledkem tedy je, že jsou k dispozici tři tabulky – faktura, transakce a vazební tabulka.

Na obrázku 4.1 je znázorněno schéma vstupních dat pomocí zjednodušeného diagramu databáze. V tomto diagramu samozřejmě nejsou zachyceny všechny tabulky ani atributy, které se v databázi nachází, nicméně jsou zde názorně vidět všechny tabulky a atributy, které jsou relevantní pro řešení dané úlohy. Zároveň tento diagram z hlediska návrhu poslouží pro znázornění interní reprezentace dat, neboť rozvržení dat do struktur `DataFrame` bude de facto kopírovat schéma těchto tabulek, tedy alespoň v úvodní fázi načítání dat před tím, než dojde na fázi jejich předzpracování.

Jak již bylo řečeno, základ tvoří 3 tabulky – `invoice`, `transaction` a jejich vazební tabulka `invoice_transaction_map`, přičemž ještě bude nutné využít tabulku `subject`, respektive atribut `ico`, neboť z hlediska použitelnosti vyvíjeného nástroje i pro třetí strany dává větší smysl použít pro identifikaci subjektů dodavatele a odběratele radši IČO nežli interní databázové ID, tudíž v rámci předzpracování budou rysy `subject_seller_id` a `subject_debtor_id` nahrazeny jejich univerzálnějším identifikátorem IČO. Vazební tabulka `invoice_transaction_map` pak obsahuje mapování některých manuálně přiřazených plateb a faktur, přičemž její využití bude spočívat zejména při generování trénovacích dat. Data z tabulek `invoice` a `transaction` pak představují hlavní část dat pro trénování i predikci. Některé atributy, jako například `recipient_message` či `variable_symbol` však nebudou využity přímo pro trénování, nýbrž poslouží pro přiřazení plateb a faktur řekněme konvenčními metodami, tedy porovnáním variabilních symbolů, či vyhledáním variabilního

symbolu faktury v poznámce pro příjemce (`recipient_message`). Výhodou tohoto přístupu bude zejména zmenšení množiny potenciálních faktur k dané konkrétní platbě, což ve svém důsledku přinese vyšší přesnost. Význam všech atributů v těchto dvou tabulkách je dle mého názoru relativně zřejmý až na jeden – specifickým atributem je `acc_id` v tabulce `transaction`. Představuje interní identifikátor účtu, na který daná transakce přišla, přičemž jeho význam spočívá opět zejména ve zmenšení množiny potenciálních faktur, neboť každá faktura má předem dáno, na který účet má platba přijít.



Obrázek 4.1: Schéma vstupních dat

4.2 Návrh algoritmu

Podstatou daného problému je de facto predikce řádků ve vazební tabulce, což ovšem principiálně není možné. Na daný problém lze ale pohlížet ještě z jiného úhlu – jako na predikci, zda pro každé možné přiřazení transakce a faktury toto přiřazení je, nebo není reálné. Predikují se tedy dvě možné hodnoty – přiřazení je reálné, nebo přiřazení není reálné, což již lze realizovat metodami strojového učení.

V rámci předzpracování tedy bude trénovací i testovací množina vytvořena následovně – nejprve se provede klasické předzpracování dat u faktur i plateb zvlášť. V dalším kroku se provede generování potenciálních kandidátů na vazbu – tedy vlastně modifikovaná operace cross-join (kartézský součin), přičemž vzhledem k efektivitě bude nutné rovnou eliminovat ty kombinace, které je možné okamžitě zařadit do kategorie nereálných kombinací – například datum splatnosti faktury je menší o více než 60 dnů oproti datu transakce, platba je starší než datum vystavení faktury, odlišné měny, a podobně. Důležité bude, aby takto vyřazených kombinací bylo co možná nejvíce, ale i tak zde hrozí vznik nevyvážených dat, se kterými se bude nutné dodatečně vypořádat. Posledním krokem v rámci předzpracování pak bude určení hodnoty cílové třídy pro nově vzniklé kombinace – to se provede velmi jednoduše tak, že kombinace je reálná, pokud existuje její příslušný záznam ve vazební tabulce mezi fakturami a transakcemi, pokud záznam neexistuje, pak kombinace není reálná.

Následuje fáze samotného trénování – zde bude nutné experimentovat s více metodami strojového učení a zvolit tu nejvhodnější na základě dosažené přesnosti.

4.3 Nasazení natrénovaného klasifikátoru

Pakliže bude dosaženo rozumné přesnosti, bude moci být natrénovaný klasifikátor nasazen v praxi. Pro každou nově příchozí platbu se nejprve vyzkouší konvenční metody, tedy například vyhledávání podle variabilního symbolu. Pokud není nalezena žádná faktura, přistoupí se k využití klasifikátoru obdobným způsobem jako při trénování – tedy nejprve je vygenerována množina potenciálních kombinací a následně je pro každou kombinaci predikována hodnota, zda je, či není taková kombinace reálná.

Kapitola 5

Implementace

V této kapitole bude podrobněji rozebrána implementační fáze. Nejprve bude popsána reprezentace dat a poté struktura aplikace jako celku. Dále pak zde načrtnu proces načítání a předzpracování dat a následně se zaměřím na rozbor použitých klasifikačních metod, se kterými budou v další fázi prováděny experimenty. Pozornost bude v menší míře věnována rovněž interpretaci výsledků predikce a jejich zobrazení uživateli, pod čímž si lze představit nápovědu pravděpodobných faktur, ke kterým se s největší pravděpodobností daná úhrada vztahuje. V neposlední řadě se nebude možné vyhnout ani popisu některých metod z výše uvedených knihoven, bez nichž by se fáze implementace v zásadě neobešla.

Aplikace byla implementována v jazyce Python verze 3.6. Pro implementaci klasifikátoru byly zvoleny již popsané knihovny, přičemž část představující prezentaci výsledků uživateli je integrována v interním systému firmy Platební instituce Roger a.s.

Vlastní implementace byla postavena zejména na informacích uvedených v dokumentacích a manuálech použitých knihoven [10, 11, 9, 14, 6, 16], krom toho pak také na ukázkách implementace z literatury [8, 1, 12, 4]. Na informacích z těchto zdrojů je tedy založen i popis implementace v této kapitole.

5.1 Reprezentace dat

Pro vnitřní reprezentaci dat je využita struktura `DataFrame` z knihovny Pandas, neboť de facto představuje abstrakci databázové tabulky a tudíž je práce s touto strukturou velmi intuitivní. Navíc díky vektorizovaným, či předkompilovaným operacím výrazně urychlí fázi předzpracování z hlediska doby běhu programu.

Každá entita, tedy zejména transakce a faktura, bude v počátku procesu předzpracování reprezentována vlastní strukturou, nicméně po předzpracování dat dojde k vytvoření jediné struktury obsahující potenciální kombinace faktur a transakcí, tedy podmnožina kartézského součinu nad těmito strukturami s eliminovanými kombinacemi, které porušují některou z nutných podmínek. Způsob vytváření těchto kombinací, jakož i eliminace těch zřejmě neplatných bude podrobněji popsána dále.

Ve fázi předzpracování navíc dojde k postupné transformaci a úpravě dat, pro trénování pak jsou data převedena do multidimenzionálního pole `ndarray` z knihovny `NumPy`.

5.2 Struktura aplikace

V této sekci je podrobně popsána struktura aplikace z hlediska rozdělení zdrojového kódu do podadresářů, včetně stručného popisu tříd. Každý z těchto podadresářů je zároveň regulérní balíček jazyka Python, takže může být importován samostatně jako modul.

Aplikace dostala pracovní název `paydentify`. V kořenovém adresáři aplikace se nachází soubor `paydentify.py`, který poskytuje veřejné rozhraní pro práci s celou aplikací pro jiné aplikace. Z hlediska jazyka Python je to tedy jediný modul, který by měl být importován, na ostatní třídy a moduly jde z hlediska ochrany přístupu nahlížet jako na typ *private*.

Následuje abecedně seřazený výčet a stručný popis hlavních podadresářů, tedy těch, jež se nacházejí v kořenovém adresáři aplikace:

- `adapters` – v tomto adresáři se nalézají adaptéry pro celou aplikaci, zejména se jedná o převod z klasického asociativního pole jazyka Python `dict`, či databázové entity na strukturu `DataFrame`.
- `classification` – zde se nacházejí třídy pro trénování (`Supervisor`), respektive predikci (`Predictor`).
- `csv_parser` – obsahuje třídy pro čtení dat ze souborů csv, zejména pro načítání trénovacích dat.
- `data` – v tomto adresáři jsou hledány csv soubory s daty pro trénování.
- `models` – sem se ve výchozím nastavení ukládají natrénované modely, nicméně cílová cesta pro uložení modelu se dá změnit argumentem příkazové řádky při spuštění skriptu na trénování dat. Dále jsou zde uloženy naučené instance tříd `OndeHotEncoder` a `RobustScaler` z knihovny `Scikit-learn`, které jsou vytvořeny při předzpracování trénovací množiny, ale při predikci je třeba instanci načítat.
- `preprocessing` – zde se nachází moduly pro předzpracování dat, a to jak faktur a transakcí, tak i modul pro sestavení trénovacího datasetu, tedy vytvoření potenciálních kombinací faktur a transakcí pomocí modifikované operace `cross-join` (kartézský součin).
- `scripts` – tento adresář obsahuje samostatně spustitelné skripty pro provádění experimentů a spouštění trénování na zvolené metody včetně uložení naučeného modelu – bez zadání parametru cílové cesty je model uložen do adresáře `models` a pojmenován dle zvolené klasifikační metody.
- `tests` – zde se nacházejí jednotkové testy modulů a tříd, u nichž je to možné, tedy těch, které mají deterministický výstup.

Kromě aplikace `paydentify`, bylo nutné provést i úpravy interní firemní aplikace firmy Platební instituce Roger a.s., která nese název `iadmin`. Struktura této aplikace je dána použitím frameworku `web2py` s drobnými úpravami, vzhledem k tomu zde tedy její strukturu popisovat nebudu a zaměřím se pouze na popis některých vyžadovaných úprav pro řešení mé diplomové práce.

5.3 Načítání dat

Pro trénování klasifikátoru jsou využita data od firmy Platební instituce Roger a.s. Data jsou uložena v produkční databázi, nicméně vzhledem k tomu, že trénování a zejména pak experimenty s jednotlivými metodami je lepší provádět lokálně, tedy mimo produkční server, je nutné relevantní data nejprve zapsat do souboru. Jako ideální formát pro tento případ se nabízí csv, navíc data lze pomocí funkcí knihovny Panda nahrávat z csv přímo do struktury `DataFrame`.

Pro predikci pravděpodobných faktur se pak jeví jako lepší nahrávat konkrétní data přímo z produkční databáze, v reakci na ad hoc požadavky od uživatelů. Zjednodušeně řečeno uživatel vybere transakce, pro které se mají nejpravděpodobnější faktury predikovat, načež budou z databáze získána potřebná data, tedy data daných transakcí a potenciálních faktur, a tyto budou následně dále převedeny do struktury `DataFrame`, na což dále navazuje proces předzpracování. Integraci a převod na `Dataframe` z různých struktur (zejména slovníků a entit) pak mají na starosti adaptéry v podadresáři `adapters`.

Načítání dat jako takové je velmi triviální díky použití knihovny Pandas, konkrétně funkce `read_csv`, kterou tato knihovna poskytuje. Tato funkce načítá data z csv souboru přímo do struktury `DataFrame`, navíc poskytuje jednu velmi důležitou výhodou – možnost použít implementaci této funkce v jazyku C pomocí argumentu `engine="c"`. Díky tomu je možné dosáhnout citelného zrychlení oproti implementaci v jazyku Python, což při načítání csv souboru se sto tisíci řádky znamená zásadní rozdíl.

5.4 Předzpracování dat

Po fázi načítání dat, je třeba tato data předzpracovat, než budou moci býti použity ke klasifikaci. Data jsou nejprve předzpracována jednotlivě, tedy každá tabulka zvlášť, načež je vytvořena množina potenciálních kombinací, tedy de facto kartézský součin s eliminací špatných kombinací. Předzpracování dat pro trénování se oproti předzpracování dat pro predikci liší jen v drobnostech, které ovšem mohou být klíčové, a proto na ně rovněž bude kladen důraz.

Kromě obligátního výběru atributů je nutno provést i předzpracování těch relevantních, zejména je nutné dát pozor na datové typy. Většina metod pro klasifikaci vyžaduje, aby všechny rysy byly zadány jako spojitý porovnatelný číselný datový typ (například `float`). U kategorických atributů je dále nutné provést kódování, k čemuž byla využita knihovna `Scikit-learn`, konkrétně třída `OneHotEncoder`, viz dále. Na závěr je pak nutné provést normalizaci.

5.4.1 Předzpracování faktur

Předzpracování faktur sestává z několika dílčích kroků – kromě obligátního výběru relevantních atributů je potřeba provést i obecné úpravy, zejména ve vztahu ke kategorickým atributům.

Předzpracování faktur je implementováno ve třídě `InvoicePreprocessor`. Vstupem této třídy jsou data z tabulky `invoice`, případně `subject` v případě potřeby nahrazení interního `id` univerzálnějším identifikátorem IČO. Formát vstupních dat byl již ilustrován v rámci návrhu na obrázku 4.1. Vstupní data musí být uložena v poli `DataFrame` – předpokládá se použití modulu `csv_parser` či implementovaných adaptérů pro integraci dat z různých zdrojů, čímž je zajištěna konzistentní struktura z hlediska interní reprezentace dat. Vý-

stupem předzpracování jsou předzpracovaná data, jež jsou rovněž uložena ve struktuře `DataFrame`.

Při procesu předzpracování faktur je nejprve provedeno zmíněné nahrazení interního databázového `id` identifikátorem IČO, což je lepší pro využití aplikace třetími stranami do budoucna.

Dále je nutné zpracovat rysy s datovým typem představující datum (`date`) – tedy rysy představující data vystavení a platnosti faktury (`issued` a `due_date`), neboť tento datový typ není vhodný pro strojové učení. Zároveň datový typ `date` není kategorický neboť hodnoty jsou porovnatelné. Nejvhodnější je dle mého názoru provést jednoduchou, ale velmi významnou úpravu – použít ordinální hodnotu daného data, tedy počet dní od počátku letopočtu.

Rysem, který je rovněž nutné předzpracovat, je variabilní symbol, který sice nebude použit přímo pro trénování, nicméně je využit pro párování faktur a plateb. Variabilní symbol je v databázi uložen jako datový typ `string` a může obsahovat chybějící hodnoty. Předzpracování je provedeno tak, že hodnoty v tomto sloupci jsou převedeny na datový typ `int64` a chybějící hodnoty jsou doplněny jako nulová hodnota, což je nutné vzít v potaz při vlastním párování. Jednak se urychlí porovnávání a zároveň je díky tomu vyřešen problém s vyhledáváním variabilního symbolu v poznámce pro příjemce, kde hodnoty `null` představují problém, neboť jsou implicitně přetypovány na `string`.

Na závěr je provedeno vyfiltrování a seřazení relevantních sloupců, načež následuje nastavení indexu a seřazení řádků, což urychlí vyhledávání a práci se strukturou `DataFrame`. Index v rámci struktury `DataFrame` představuje jednoznačný identifikátor řádků, z tohoto pohledu se tedy jedná o obdobu pojmenování sloupců jmény atributů, což rovněž zajišťuje jejich identifikaci. Jako index je zde použit identifikátor `id`, který nebude využit přímo ke trénování, ale je nutno jej zachovat jako identifikaci konkrétní faktury pro reprezentaci výsledků predikce.

5.4.2 Předzpracování transakcí

Implementace předzpracování je dána třídou `TransactionPreprocessor`, vstupem je struktura `DataFrame` obsahující data transakcí získaná z adaptérů, či csv souborů. Výstupem je opět strutura `DataFrame` s předzpracovanými relevantními daty.

Vlastní předzpracování transakcí je v mnohém podobné předzpracování faktur, až na některé drobné rozdíly. V první řadě je nutno vybrat pouze příchozí transakce na konkrétní daný účet. Datum a variabilní symbol jsou předzpracovány zcela totožným způsobem, jako je tomu u faktur. Nový atribut oproti fakturám, který je třeba rovněž zpracovat, je bankovní účet. Jedná se o kategorický atribut, což znamená, že je třeba jej zakódovat, nicméně tento krok bude proveden až po vytvoření trénovací množiny. Co je ale nutné provést hned je sjednocení formátu bankovního účtu, zejména co se týká použití kódu banky, nebo BIC. Výsledkem je, že jeden bankovní účet by měl představovat jednu kategorickou hodnotu.

5.4.3 Sestavení potenciálních kombinací

Po předzpracování dat faktur a transakcí samostatně je nutné sestavit jejich potenciální kombinace, pod čímž si lze představit zejména kartézský součin s eliminací těch kombinací, jež porušují některou z podmínek. Postup tvorby těchto kombinací zde bude popsán podrobněji, neboť je klíčovým při řešení daného problému.

Sestavení potenciálních kombinací je implementováno ve třídě `DatasetBuilder`. Vstupem jsou struktury `DataFrame` s předzpracovanými daty faktur a transakcí, nepovinným

parametrem je pak mapování `invoice_transaction_map`, které má využití při sestavování množiny dat pro trénování.

Samotné sestavení pak probíhá způsobem jenž byl již naznačen v návrhu – datová množina je vytvořena pomocí operace cross-join (kartézský součin) s eliminací těch kombinací, jež porušují některou z podmínek pro to, aby daná kombinace mohla být považována za validní, například platba je provedena před vystavením faktury a podobně. Samozřejmě vzhledem k efektivitě a zejména paměťové složitosti se toto neděje přesně v tomto pořadí a z toho důvodu není využita přímo operace cross-join. Kombinace, o kterých lze předem říct, že nejsou validní, ani nejsou vytvořeny, vznikne tedy de facto podmnožina kartézského součinu transakcí a faktur, nicméně výsledek je totožný.

Podmínkami, které určují, zda daná kombinace může být validní, jsou dány zejména logickým vztahem mezi daty vystavení či splatnosti faktury a datem uskutečnění platby. Další poněkud specifickou podmínkou, která platí v prostředí dat firmy Platební instituce Roger a.s., nicméně obecně zřejmě platit nemusí, je pak požadavek na rovnost měn faktury a transakce. Pro shrnutí jsou tedy rozhodující podmínky tyto:

- Datum vystavení faktury je menší než datum provedení platby.
- Datum provedení platby je menší než datum splatnosti s tolerancí 30 dnů.
- Měna na faktuře odpovídá měně platby.

Princip sestavení je tedy relativně prostý – nejprve jsou pro každou fakturu nalezeny potenciální transakce, načež je zavolána metoda `merge` třídy `DataFrame`. Na závěr je ještě nutné změnit index tak, aby byl hierarchický a obsahoval identifikátory faktury a transakce pro danou kombinaci, pro což se skvěle hodí třída `MultiIndex` z knihovny `Pandas`. Tento krok je vyžadován zejména k tomu, aby po provedení predikce bylo možné vrátit výsledek ve formě mapování identifikátorů faktur a transakcí, což by při ztrátě této informace možné nebylo, a zároveň tyto identifikátory nesmí být ponechány přímo v datové množině, neboť by negativně ovlivnily trénování.

Cílová třída

Specifikem, kterým se sestavení potenciálních kombinací pro trénovaní odlišuje od verze pro predikci, je pak sestavení atributu cílové třídy. Jak již bylo naznačeno v návrhu, hodnoty cílové třídy mohou nabývat dvou hodnot – jedná se tedy o binární klasifikaci. Hodnota 1 značí, že daná kombinace je správná, hodnota 0 pak představuje kombinaci neplatnou. Hodnota tohoto atributu je doplněna buď na základě přítomnosti identifikátorů faktury a transakce dané kombinace v tabulce `invoice_transaction_map`, nebo na základě rovnosti variabilních symbolů.

I přes to, že tímto způsobem sestavení potenciálních kombinací vzniknou velmi nevyvážená data, tj. míra zastoupení třídy 1 bude hrubě menší než míra zastoupení třídy 0, je tento postup nezbytný, vzhledem k tomu, že predikovat přímo identifikátory je principiálně nemožné. V rámci předzpracování, by zde za normálních okolností ještě následovalo vyvážení dat například podvzorkováním, nicméně během vývoje se ukázalo, že lepším přístupem pro vypořádání se s nevyváženými daty bude nastavením vah cílovým třídám, což bude podrobněji popsáno dále.

5.4.4 Transformace

Následně je nutné spočítat nové rysy, které by mohly posloužit ke klasifikaci kombinací. V rámci této fáze zaniknou některé rysy a vzniknou nové, technicky jde tedy z hlediska předzpracování o fázi transformace dat, konkrétně o konstrukci rysů. Výpočet nových atributů není nijak zvlášť obtížný, velkou výhodou je použití knihovny `Pandas` a vektorizovaných operací, čímž se doba této fáze velmi urychlí. Vlastnosti, které by principiálně mohly pomoci jsou tyto:

- Podíl částek – první z nových rysů pro klasifikaci kombinací faktur a transakcí je podíl částek. Výpočet tohoto nového atributu je vcelku zřejmý, spočte se jako podíl nominální hodnoty faktury a zaslané úhrady. Možná se nabízí otázka, proč zde místo podílu radši nepoužít rozdíl – důvodem je, že platby i transakce se v rámci trénovací množiny vyskytují v různých měnách, a tudíž by rozdíl mohl být zavádějící.

- Doba od vystavení faktury – tedy de facto rozdíl mezi datem vystavení faktury a datem uskutečnění platby

- Doba do splatnosti faktury – která se spočítá triviálně jako rozdíl mezi datem splatnosti a datem uskutečnění platby. Zde jen platí drobné specifikum a to že faktura může být uhrazena až po splatnosti. Nicméně záporná čísla z hlediska klasifikace problém nepředstavují, neboť hodnoty atributů budou stejně v následujícím kroku normalizovány.

- Vztah bankovního účtu a odběratele – tento nový rys je kategorického typu, tudíž je nutné provést konverzi. K tomu je využita třída `OneHotEncoder` z knihovny `Scikit-learn`. Instance této třídy je po natrénování uložena tak, aby mohla být znovu použita pro predikci – k uložení instance je využita standardní knihovna `pickle`. Výstupem zakódování kategorického atributu touto metodou je řídký vektor určující příslušnost k jedné z kategorií – tedy laicky řečeno jeden z prvků vektoru je roven jedné, zbytek jsou nuly.

Tyto nové rysy budou přímo využity pro trénování a predikci, ostatní zanikají – výjimku tvoří identifikátory transakce a faktury dané kombinace, které však nejsou v rámci struktury `DataFrame` uloženy přímo jako sloupec, nýbrž jako hierarchický index (`MultiIndex`). Výjimku představuje rovněž atribut cílové třídy, nicméně ten se logicky vyskytuje pouze u trénovací množiny.

5.4.5 Normalizace

Jako další fáze předzpracování dat je provedena normalizace. Tento krok je velmi důležitý, neboť některé metody vyžadují, aby data byla rozložena normálním Gaussovým rozdělením kolem nuly s jednotkovým rozptylem. Tento krok je nutno provést pro každý z atributů.

V rámci vývoje byly provedeny experimenty se dvěma odlišnými přístupy normalizace, které poskytuje knihovna `Scikit-learn`, a to třídy `StandardScaler` a `RobustScaler`. Vzhledem k názvu zde může vyvstat nejasnost mezi anglickými termíny *scale* a *normalize*, které se i v literatuře často zaměňují, nicméně v kontextu knihovny `Scikit-learn` [13] je pod pojmem normalizace myšlena normalizace vektoru na jednotkovou normu. Normalizace z hlediska rozdělení se zde označuje termíny *standarization*, či *scaling*, avšak pro zachování konzistence se dále budu spíše držet názvosloví z knihy *Data Mining: Concepts and Techniques*[3].

Výhodou využití knihovny `Scikit-learn` je zejména fakt, že vstupem zde je pole `ndarray` potažmo `DataFrame`, tedy všechny atributy jako jedno dvoudimenzionální pole. Výpočet tak je vektorizován nad celým tímto polem, tedy laicky řečeno nejen po řádcích, ale i po sloupcích, což vede k výraznému zrychlení. Výstupem jsou pak normalizovaná data v poli `ndarray`.

Principem metody, kterou implementuje třída `StandardScaler` je tzv. z-skóre, výpočet je dán vzorcem:

$$z = \frac{x - \mu}{\sigma}$$

kde μ představuje střední hodnotu a σ směrodatnou odchylku – ty jsou spočítány na základě dat trénovací množiny. Jedná se asi o nejčastěji používanou metodu, nicméně nevýhodou tohoto přístupu je zejména citlivost vůči odlehlým hodnotám, což se ukázalo jako rozhodující faktor.

Oproti tomu metoda, jež je implementována knihovnou `RobustScaler` funguje na principu mezikvartilového rozpětí (IQR), díky čemuž je odolná vůči výskytu odlehlých hodnot. Tato metoda byla nakonec využita ve finální verzi implementace, neboť dosahovala mnohem lepších výsledků.

Hodnoty pro normalizaci dat jsou získány při procesu trénování a je nutné je nějakým způsobem ukládat tak, aby mohly být znovu využity pro normalizaci dat k predikci. Vzhledem ke konzistenci z hlediska implementace se jeví jako nejlepší řešení opět využít knihovny `pickle`, pomocí které je celá instance dané třídy (v tomto případě třídy `RobustScaler` serializována a uložena. Naproti tomu při predikci už stačí jen danou instanci načíst ze souboru a deserializovat a následně použít k normalizaci dat.

5.4.6 Nevyváženost dat

Vypořádání se s nevyvážeností dat představuje fázi specifickou pouze pro trénovací množinu. Jak již bylo řečeno, data, která vzniknou popsaným způsobem, jsou velmi nevyvážená – zastoupení majoritní třídy představující nevalidní kombinaci faktury a transakce je více než stokrát větší než zastoupení třídy minoritní, která představuje kombinaci validní.

Obecně existuje více možností řešení, v rámci vývoje bylo experimentováno se dvěma odlišnými přístupy. Prvním z nich je podvzorkování, což je v podstatě náhodné odstranění některých vzorků náležejících k majoritní třídě. Druhým pak je nastavení vah cílovým třídám podle podpory v trénovacích datech. Experimentálně se ukázalo, že lepších výsledků dosahuje použití druhého způsobu.

Z hlediska implementace probíhá nastavení vah jednotlivým cílovým třídám velmi jednoduše – rozdělení vah se předá klasifikátoru jako parametr při jeho instanciaci. Nevýhodou však je, že ne všechny klasifikátory z knihovny `Scikit-learn` možnost nastavení vah jednotlivým třídám podporují. K určení správných vah nabízí knihovna `Scikit-learn` funkci `compute_class_weight`, výpočet je proveden na základě vzorce:

$$w_i = \frac{N_{samples}}{N_{classes} \times N_i}$$

kde $N_{samples}$ je počet vzorků v trénovací množině, $N_{classes}$ počet všech tříd a N_i podpora jedné konkrétní třídy C_i, tedy počet vzorků z trénovací množiny, jež náleží do třídy C_i. Výpočet je opět urychlen využitím vektorizovaných operací knihovny `NumPy`.

5.5 Trénování klasifikátoru

Další část implementace pak představuje trénování klasifikátoru, což je realizováno třídou `Supervisor`. Implementaci tvoří převážně experimenty s různými metodami klasifikace. K tomu byly využity zejména knihovny `Scikit-learn` a `Keras`.

Vlastní trénování klasifikátoru nepředstavuje z hlediska implementace až tak zajímavou část, neboť se jedná de facto jen o instanciaci dané třídy klasifikátoru z uvedených knihoven a zavolání metody `fit`. Zajímavější však jsou experimenty a srovnání výsledků, o čemž je ale pojednáváno v jiné kapitole.

Natrénované klasifikátory je nutné ukládat, aby mohly být případně znovu použity u predikce, k čemuž je opět využita serializace pomocí knihovny `pickle`. Výjimku představuje ukládání neuronové sítě implementované knihovnou `Keras`, které je dle doporučení realizováno ve formátu HDF5.

Pro podporu experimentování byla dále využita metoda zvaná *grid search*, což je v podstatě hledání optimálního nastavení parametrů hrubou silou. Knihovna `Scikit-learn` nabízí již existující implementaci `GridSearchCV`. Nevýhodou může být fakt, že vzhledem k tomu, že jde o optimalizaci hrubou silou, je nutno počítat s dlouhou dobou hledání optimálního nastavení, zejména u klasifikátorů, u nichž je doba trénování relativně dlouhá. Zároveň platí, že některé parametry se vzájemně ovlivňují, tudíž u nich moc nedává použití metody *grid search* smysl.

5.6 Predikce

Další část implementace pak představuje prediktor validity daných kombinací faktur a transakcí, který je realizován třídou `Predictor`. Vstupem jsou předzpracovaná data, u kterých však příslušnost k cílové třídě určující validitu není předem známá. Výstupem je pak slovník, který pro každou transakci obsahuje seznam faktur, k nimž se daná transakce pravděpodobně vztahuje. Tento slovník je sestaven na základě identifikátorů faktur a transakcí, které však v rámci struktury `DataFrame` nejsou uloženy přímo jako atributy (sloupce), nýbrž představují hierarchický vícenásobný index (`MultiIndex`).

Samotnou predikci pak lze rozdělit na několik fází. Nejprve je natrénovaný klasifikátor načten ze souboru. Kromě toho musí být ze souborů načteny i třídy `OneHotEncoder` a `RobustScaler`, jak bylo popsáno v sekci o předzpracování. Po provedení příslušných transformací, které zajistí normalizaci dat do podoby, na kterou byl klasifikátor trénován je možno povést vlastní predikci, což představuje pouhé zavolání jedné metody klasifikátoru – `predict`, která pro vstupní data vrátí pole `ndarray` predikovaných hodnot. Toto pole je tvořeno predikovanými hodnotami nula nebo jedna, kdy 0 představuje nevalidní kombinaci a 1 kombinaci validní. Výjimku představuje pouze neuronová síť definovaná pomocí knihovny `Keras`, která vrátí dosažené skóre, tedy reálné číslo mezi 0 a 1, tudíž je nutné ještě stanovit hranici (*threshold*) pro rozdělení do tříd – vzhledem k tomu, že v našem případě existují dvě cílové třídy, je touto hranicí hodnota $0,5$.

Na závěr je nutné provést interpretaci výsledků predikce, tedy určení mapování faktur a transakcí a převod na klasický slovník (`dict`) jazyka Python. Příslušné metody k tomu nabízí přímo třída `DataFrame` – nejprve je zavolána metoda `groupby` pro seskupení dat dle identifikátorů transakcí, následně jsou identifikátory faktur, které tvoří samostatnou hierarchii indexu, v každé takto seskupené množině převedeny na seznam (`list`). A konečně – celá struktura `DataFrame` je převedena na slovník. Výsledný slovník tedy obsahuje jako klíče identifikátory transakcí, hodnoty pak tvoří seznamy předpokládaných faktur, což

je formát, který je dobře zpracovatelný jakoukoliv aplikací jazyka Python, bez nutnosti používat některé knihovny navíc.

5.7 Integrace do interního systému

Výslednou aplikaci bylo nutné integrovat do interního firemního systému tak, aby mohly být výsledky predikce interpretovány uživatelům. Z hlediska návrhu je vše koncipováno tak, že rozhraní pro práci s aplikací poskytuje třída `Paydentify` tak, aby práce byla co nejjednodušší. De facto je tedy nutné pouze nahrát data transakcí a faktur a následně zavolat metodu `predict`, která vrátí slovník obsahující jako klíče identifikátory transakcí a jako hodnoty seznamy předpokládaných faktur, jak bylo popsáno v předchozí sekci. Implementace získání dat z databáze a prezentace výsledků uživateli pak byla implementována v interním firemním systému *iadmin*.

Nyní bude celý proces popsán trochu podrobněji. Nejprve je nutné získat data faktur a transakcí z databáze – z hlediska predikce dává smysl vyfiltrovat pouze příchozí transakce dle typu účtu a dosud nezaplacené faktury, které by mohly být platbou na daný účet zaplaceny – ať už zcela, nebo jen částečně. Následně jsou data předány prediktoru – třída `Paydentify` podporuje nahrávání dat pomocí adaptérů nejen ve struktuře `DataFrame`, ale i v klasickém slovníku jazyka Python, nebo jako specifickou entitu, s níž se v rámci interních firemních aplikací pracuje. Dalším krokem pak je zavolání metody `predict` ze třídy `Paydentify`, které automaticky provede i veškeré předzpracování dat, což je pro jednoduchost z hlediska použití aplikace `paydentify` velmi důležité. Tato metoda zároveň vrací výše popsaný slovník mapování faktur a transakcí, což de facto znamená, že tato metoda predikuje přímo vazby mezi fakturami a transakcemi a nutnost vytvářet potenciální kombinace s následnou klasifikací je z pohledu ostatních interních aplikací skryta.

Na konec následuje fáze prezentace predikovaných faktur uživateli. Uživatelské rozhraní pro tento úkol bylo z velké části převzato z již existujících částí aplikace *iadmin*, tudíž zde nebudu zabíhat do přílišných detailů. První část uživatelského rozhraní je realizována pomocí tabulky transakcí, pro výběr transakce pro predikci. Po kliknutí na danou transakci se zobrazí seznam pravděpodobných faktur, jejichž úhradu by tato transakce mohla představovat – může samozřejmě představovat úhradu více faktur zároveň. Každá faktura v tomto seznamu je pak odkaz na detail faktury, kde jsou jednak zobrazeny všechny detailní údaje dané faktury, ale zároveň tato stránka slouží i pro provedení akcí spojených s uhrazením faktur.

Kapitola 6

Experimenty a testování

V této kapitole budou probrány experimenty s některými vybranými klasifikačními metodami, včetně nastavení jejich parametrů. Experimenty vždy probíhají klasickým způsobem, kdy nejprve je daný klasifikátor natrénován pomocí trénovacích dat, a následně je pomocí různých metrik otestována a vyhodnocena jeho přesnost na testovacím datasetu. Obecně platí, že množiny trénovacích a testovacích dat mají prázdný průnik.

V této kapitole budou popsány konkrétně nejen jednotlivé klasifikační metody a jejich parametry, ale i metriky pro vyhodnocení spolehlivosti. Pro vykreslení grafů byly využity informace a ukázky implementace z tutoriálu na klasifikaci nevyvážených dat [2]. Pro výběr vhodných metrik pak byly informace čerpány ze článku o vlivu nevyváženosti dat na některé nejznámější metriky [5].

6.1 Výběr vhodných metrik

Prvním úkolem bude volba vhodných metrik pro porovnání výsledků jednotlivých metod. Vzhledem k velké nevyváženosti jak trénovací, tak testovací množiny nebude základní metrika *accuracy* dostačující, neboť jelikož je nevyváženost více než stonásobná, prostou predikcí všech vzorků do majoritní třídy by bylo touto metrikou dosaženo skóre 99% a víc.

Podle autorů článku [5] se po provedení experimentů zkoumajících dopad nevyváženosti na výsledky metrik jako nejodolnější ukázala metrika AUC-ROC, která dokonce jako jediná ze zkoumaných metrik prakticky nebyla nevyvážeností ovlivněna. Na základě toho bude tato metrika zvolena jako směrodatná, nicméně pro srovnání zde budou uvedeny i výsledky jiných metrik pro jednotlivé klasifikační metody. Princip těchto metrik byl již popsán v teoretické části této diplomové práce. Zároveň bude u každé metody uvedena i výsledná matice záměn.

Z hlediska praktického využití prediktoru validity kombinací, tedy pro identifikaci plateb, je pak poměrně důležitou metrikou i *recall*. To je dáno tím, že systém by měl umět nabízet k dané platbě některé nejpravděpodobnější faktury, přičemž zde je výhodnější, když se v nabídce s velkou pravděpodobností objeví většina faktur i za cenu vyšší míry *false positive*, tedy s nižší hodnotou *precision*. Pokud by metrika *recall* pro minoritní třídu dosahovala jen nízkých hodnot, pak by to de facto znamenalo, že velké množství ve skutečnosti validních kombinací nebude do nabídky vůbec zahrnuto.

6.2 Výsledky experimentů

V této sekci budou prezentovány výsledky provedených experimentů pro jednotlivé klasifikační metody. U všech experimentů byla data rozdělena na trénovací a testovací množiny v poměru 3:1, tedy trénovací množina představovala 75% a testovací množina 25% všech potenciálních kombinací faktur a transakcí vytvořených z dat, jež byly dodány firmou Platební instituce Roger a.s. pro účely mé diplomové práce. Z celkového počtu 107 252 kombinací faktur a transakcí tak trénovací množinu tvořilo 80 439 kombinací a testovací množinu 26 813 kombinací. Zároveň je samozřejmě dodržena podmínka prázdného průniku trénovací a testovací množiny a celá datová množina byla před rozdělením náhodně promíchána.

Kromě experimentů s různými metodami byly provedeny i experimenty s nastavením parametrů, k čemuž byla využita již popsaná metoda *grid search*. Pro měření a vyhodnocení výsledků jednotlivých klasifikátorů tedy byly použity nejlepší parametry získané pomocí této metody.

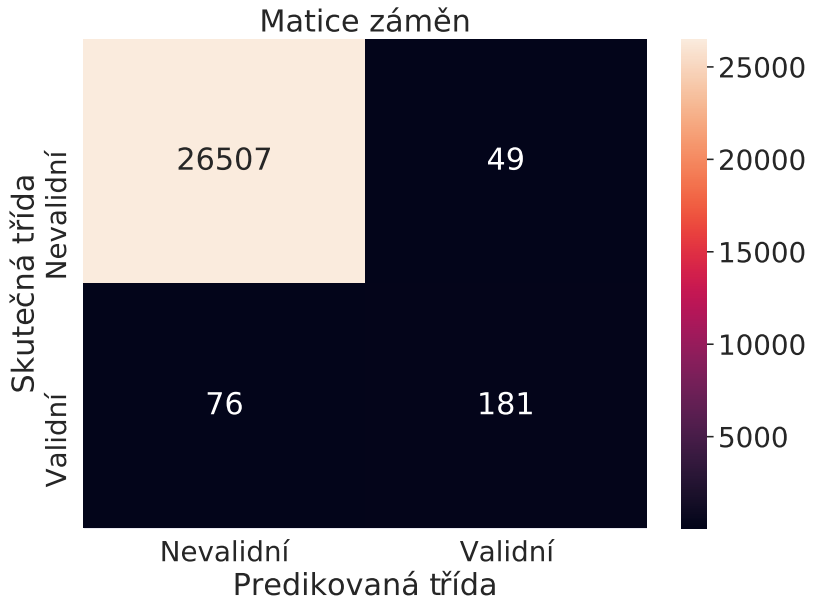
Pro výpočet vybraných metrik je využita funkce `classification_report` z knihovny `Scikit-learn`, která ale neobsahuje výpočet metriky AUC-ROC vzhledem k tomu, že tato metrika vyžaduje navíc výpočet pravděpodobnosti. Tato metrika je tedy dopočítána dodatečně pomocí funkce `roc_auc_score`. Výpis funkce `classification_report` obsahuje výsledek metrik pro jednotlivé třídy, načež jsou spočítány vážené a nevážené průměry. Na nevyvážených datech dosahuje samozřejmě nižších hodnot nevážený průměr.

Tabulky obsahující výsledky vybraných metrik u každé z metod budou přibližně odpovídat formátu výpisu funkce `classification_report` s přidanou výslednou hodnotou metriky AUC-ROC. U metriky AUC-ROC není nutné průměrné hodnoty počítat, neboť představuje pravděpodobnost, že náhodně vybraný pozitivní vzorek bude mít vyšší skóre, než náhodně vybraný negativní vzorek – výsledkem je tedy, alespoň v případě binární klasifikace, jedna stejná hodnota nezávisle na vyváženosti či nevyváženosti dat, díky čemuž je právě tato metrika vůči nevyváženosti odolná, jak bylo experimentálně potvrzeno v článku [5].

6.2.1 Metoda k-nejbližších sousedů

Pro klasifikační metodu k-nejbližších sousedů (KNN) byla využita implementace z knihovny `scikit-learn` – `KNeighborsClassifier`. Parametrem této metody je zejména počet sousedů – pomocí metody *grid search* byla ze zkoumaných hodnot 5, 10, 15 a 20 nalezena jako nejlepší hodnota 5, nicméně nutno podotknout, že výsledky všech hodnot tohoto parametru byly relativně stejné. Nevýhodou této metody je zejména absence parametru pro nastavení vah jednotlivým třídám, což se projevuje u výsledků vyhodnocení. Dále obecně pro tuto metodu platí, že není efektivní pro data s větším počtem dimenzí.

Na obrázku 6.1 je znázorněna výsledná matice záměn pomocí tzv. *heat mapy*. Třída *nevalidní* představuje kombinace faktur a transakcí, jež mezi sebou nemají vazbu, třída *validní* pak představuje validní kombinaci jedné faktury a jedné transakce, které mezi sebou vazbu mají. Je vidět, že úspěšnost predikce majoritní třídy, tedy hodnoty *true negative*, jsou relativně dobré, o poznání hůř je na tom predikce minoritní třídy.



Obrázek 6.1: Matice záměn metody k-nejbližších sousedů

Tabulka 6.1 pak ukazuje výsledky vybraných metrik zaokrouhleny na 2 desetinná místa. Třída 0 představuje třídu nevalidních kombinací, třída 1 pak třídu kombinací validních. Hodnota *support* představuje podporu tj. zastoupení jednotlivých tříd v rámci testovací množiny. Toto rozložení odpovídá nevyváženosti trénovací množiny. Následují hodnoty metrik *precision*, *recall* a jejich harmonického průměru – f_1 score. Zde je pozorovatelný rozdíl ve výsledcích jednotlivých tříd, kdy majoritní třída je predikována relativně spolehlivě, predikce třídy minoritní už vykazuje nepřesnosti. Hodnota *macro avg* pak představuje jejich nevážený průměr, oproti tomu hodnota *weighted avg* představuje průměr vážený na základě podpory jednotlivých tříd. Vzhledem k nevyváženosti dat je však v tomto ohledu směrodatnější spíše průměr nevážený, který samozřejmě dosahuje horších výsledků. Pro úplnost následuje hodnota metriky *accuracy*, kterou výpis funkce `classification_report` rovněž obsahuje, nicméně jak již bylo řečeno, její vypovídací hodnota není v případě nevyvážených dat příliš vysoká. Na závěr je pak uvedena metrika AUC-ROC, ta v tomto případě dosahuje překvapivě relativně slušných hodnot.

Pro shrnutí tedy metoda KNN nevykazuje úplně špatné výsledky zejména pokud jde o predikci nevalidních kombinací, nicméně vzhledem relativně nízkým hodnotám metriky *recall* u třídy validních kombinací není pro praktické využití k identifikaci plateb příliš vhodná.

	support	precision	recall	f1-score
0	26556	1,00	1,00	1,00
1	257	0,79	0,70	0,74
macro avg		0,89	0,85	0,87
weighted avg		1,00	1,00	1,00
accuracy			1,00	
AUC-ROC			0,95	

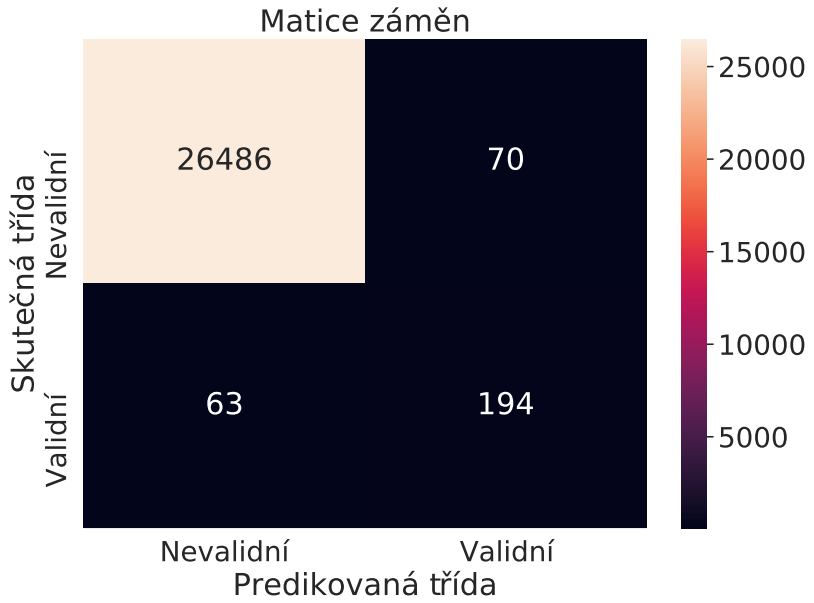
Tabulka 6.1: Výsledky vybraných metrik pro metodu KNN.

6.2.2 Rozhodovací strom

Pro implementaci klasifikátoru založeném na rozhodovacím stromu byl využit klasifikátor `DecisionTreeClassifier` z knihovny `scikit-learn`. Velkou výhodou zde je možnost zadat jako parametr klasifikátoru nastavení vah cílových tříd na základě podpory, ostatní parametry byly ponechány ve výchozím nastavení. Za další výhodu tohoto typu klasifikátoru by se pak dala označit i relativně nízká doba trénování, což kromě urychlení experimentů poskytuje i možnost seskupit a trénovat více klasifikátorů tohoto typu (metoda textitrandom forest), aniž by to vedlo k velmi dlouhé době trénování.

Výsledná matice záměn je opět znázorněna pomocí heat mapy viz obrázek 6.2, výsledky jsou relativně srovnatelné s předchozí metodou KNN, pouze se mírně snížila míra *true negative* ve prospěch míry *true positive*, zároveň s ní se ovšem zvýšila i chybová míra *false positive*.

Tabulka 6.2 výsledků vybraných metrik pak dodržuje formát zavedený u předchozí metody. Je vidět, že výsledky jsou velmi podobné výsledkům přechozí metody KNN, nicméně se zvýšila hodnota metriky *recall*, tudíž pro praktické využití by byla metoda rozhodovacího stromu lepší, nežli metoda KNN. Negativní věcí však je, že se poměrně dost snížila hodnota metriky AUC-ROC, což může znamenat skrytý problém v hodnotách skóre predikovaného touto metodou.



Obrázek 6.2: Výsledná matice záměn pro rozhodovací strom

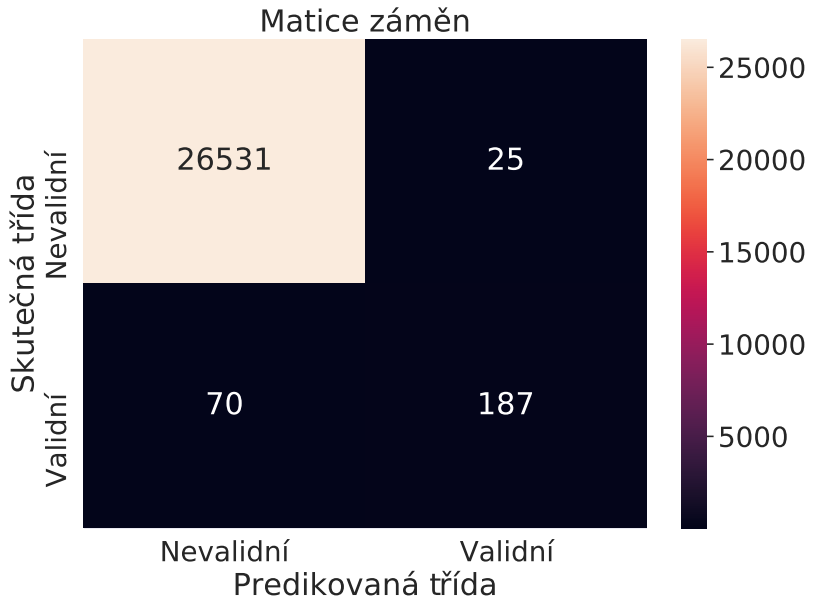
	support	precision	recall	f1-score
0	26556	1,00	1,00	1,00
1	257	0,73	0,75	0,74
macro avg		0,87	0,88	0,87
weighted avg		1,00	1,00	1,00
accuracy			1,00	
AUC-ROC			0,88	

Tabulka 6.2: Výsledky vybraných metrik pro rozhodovací strom.

6.2.3 Náhodný les

Náhodný les je soubor rozhodovacích stromů, přičemž jako výsledná třída je zvolen modus, jedná se tedy ve své podstatě o hlasování. Implementace této metody je realizována třídou `RandomForestClassifier` z knihovny `scikit-learn`, která umožňuje nastavení vah jednotlivým třídám dle zastoupení. Důležitým parametrem je počet rozhodovacích stromů, pro který byla v rámci experimentů použita výchozí hodnota 100.

Obrázek 6.3 zobrazuje heat mapu výsledné matice záměn. Z výsledků je možno vyčíst, že tato metoda vykazuje oproti předchozím vyšší míru *true negative* a nižší *false positive*, ovšem míra *false negative* je pořád relativně vysoká.



Obrázek 6.3: Výsledná matice záměn pro náhodný les

Tabulka 6.3 prezentuje výsledky vybraných metrik pro tuto metodu. Celkově se dá říct, že výsledky všech metrik se oproti experimentům s předchozími metodami zvýšily, bohužel až na metriku *recall*, která jak bylo zmíněno je v tomto případě z hlediska praktického využití velmi podstatná. Výsledek metriky *accuracy* je po zaokrouhlení na dvě desetinná místa opět roven 1, nicméně jak již bylo zmíněno, tato metrika není vzhledem k nevyváženosti úplně relevantní. Ovšem metrika AUC-ROC dosahuje pro tuto metodu velmi slušné hodnoty 0,97, což už vypovídací hodnotu má. Při srovnání výsledných hodnot pro tento klasifikátor typu náhodný les s předchozími se dá tento označit jako doposud nejúspěšnější.

	support	precision	recall	f1-score
0	26556	1,00	1,00	1,00
1	257	0,88	0,73	0,80
macro avg		0,94	0,86	0,90
weighted avg		1,00	1,00	1,00
accuracy			1,00	
AUC-ROC			0,97	

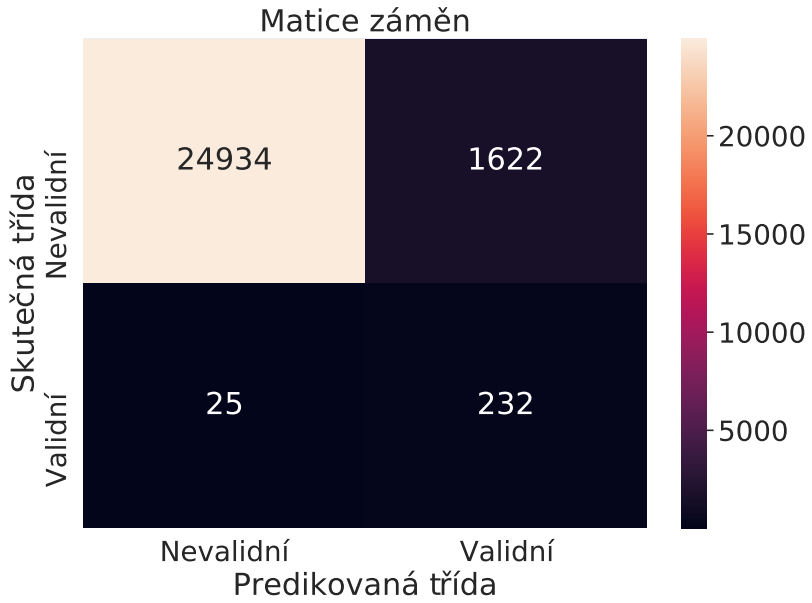
Tabulka 6.3: Výsledky vybraných metrik pro klasifikátor typu náhodný les.

6.2.4 SVM

Pro implementaci metody *Support vector machines* (SVM) byl zvolen klasifikátor realizovaný třídou `LinearSVC` z knihovny `scikit-learn`. Jedná se o tzv. *C-support vector* variantu klasifikátoru SVM. Experimenty pomocí metody *grid search* byly prováděny se třídou `SVC` a jako optimální se ukázala lineární kernelová funkce, kvůli čemuž byla zvolena varianta této třídy `LinearSVC`. Pro regularizační parametr `C` pak byla jako nejlepší zjištěna hodnota 10. Nevýhodou této metody je relativně dlouhá doba trénování pro větší množství vzorků v trénovací množině, což je i náš případ.

Z matice záměn na obrázku 6.4 lze vypozorovat, že míra *false negative* je velmi nízká, nicméně za cenu rapidního zvýšení míry *false positive*. Míra *true positive* je pak s hodnotou 232 z doposud popsaných metod zatím nejvyšší.

V tabulce 6.4 pak jde vidět hodnoty jednotlivých vybraných metrik pro tutu klasifikační metodu. Ve shodě s očekáváním po přezkumu hodnot v matici záměn je míra *recall* velmi slušná, na druhou stranu *precision* pro minoritní třídu dosahuje jen hodnoty 0,13. Metrika AUC-ROC pak vykazuje hodnotu 0,96 což je relativně slušné. I přes velmi dobrou hodnotu metriky *recall* se však vzhledem k dosti špatné hodnotě u metriky *precision* tento klasifikátor nejeví jako příliš využitelný pro praktické účely, neboť množství predikovaných potenciálních faktur by pravděpodobně bylo až příliš vysoké, což by přínos prediktoru velmi snížilo. Potenciálně by však tento klasifikátor mohl být využit ve spojení s nějakým dalším klasifikátorem, například pomocí hlasovacího mechanismu, či metody *AdaBoost*.



Obrázek 6.4: Výsledná matice záměn pro metodu SVM

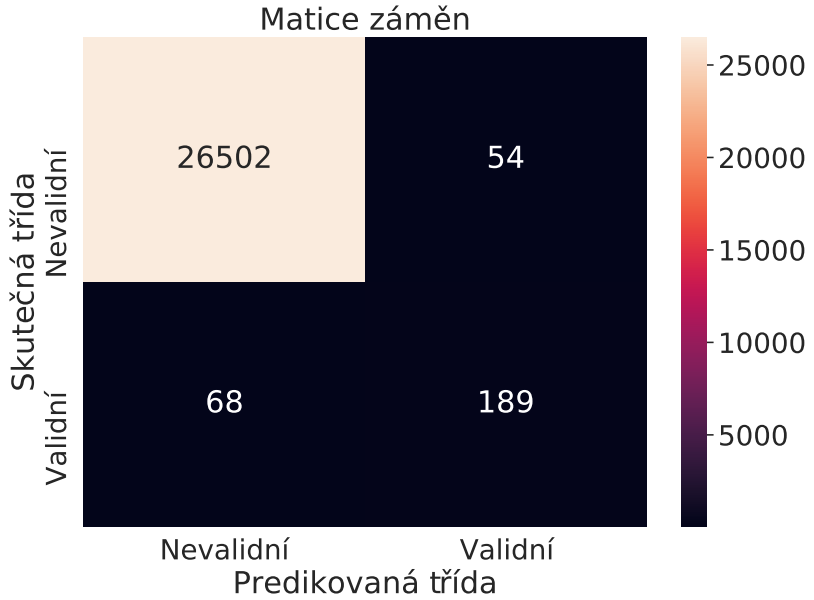
	support	precision	recall	f1-score
0	26556	1,00	0,94	0,97
1	257	0,13	0,90	0,22
macro avg		0,56	0,92	0,59
weighted avg		0,99	0,94	0,96
accuracy			0,94	
AUC-ROC			0,97	

Tabulka 6.4: Výsledky vybraných metrik pro SVM klasifikátor

6.2.5 Neuronová síť MLP pomocí knihovny `scikit-learn`

Další metoda, se kterou byly provedeny experimenty, pak je neuronová síť, konkrétně vícevrstvá síť perceptronů (MLP), implementovaná pomocí knihovny `scikit-learn` třídou `MLPClassifier`. Nevýhodou tohoto klasifikátoru je kromě relativně dlouhé doby trénování, což ale platí pro neuronové sítě obecně, zejména nemožnost zadat parametrem nastavení vah cílovým třídám. Jako aktivační funkce byla zvolena funkce *ReLU* (*rectified linear unit*), která je standardně definována jako $f(x) = max(x, 0)$jako optimalizátor pak algoritmus *Adam*[7].

Matice záměn na obrázku 6.5 zachycuje predikci na testovací množině ve srovnání s realitou. Výsledky nijak zásadně nevybočují z řady oproti předchozím experimentům.



Obrázek 6.5: Matice záměn pro metodu MLP pomocí knihovny `scikit-learn`

V tabulce 6.5 je pak možno nalézt výsledky vybraných metrik. V zásadě se dá říct, že výsledky se víceméně neliší od předchozích zkoumaných metody (vyjma SVM), pouze metrika AUC-ROC dosahuje velmi slušné hodnoty 0,98. Vzhledem k nízké míře *recall* však nejspíše ani tato metoda neposkytne praktické využití při identifikaci plateb.

	support	precision	recall	f1-score
0	26556	1,00	1,00	1,00
1	257	0,78	0,74	0,76
macro avg		0,89	0,87	0,88
weighted avg		1,00	1,00	1,00
accuracy			1,00	
AUC-ROC			0,98	

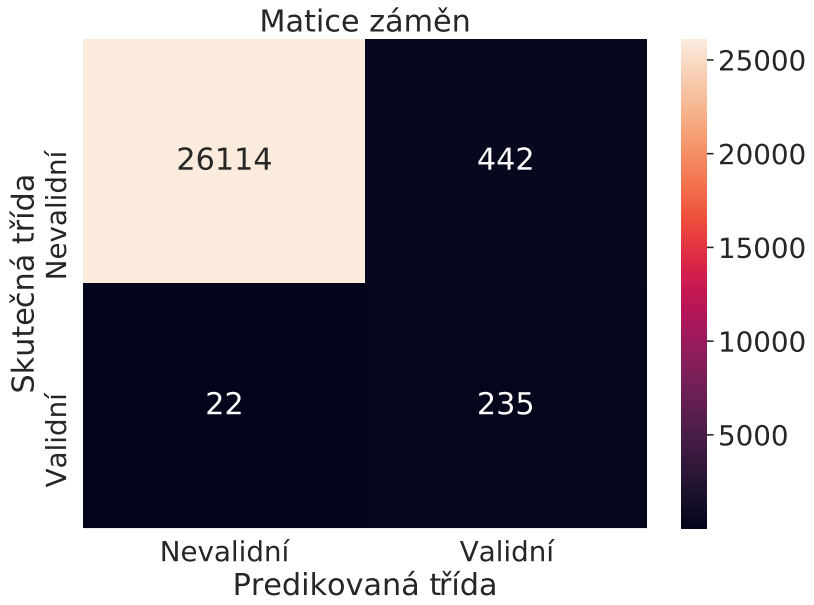
Tabulka 6.5: Výsledky metrik pro klasifikátor MLP z knihovny `scikit-learn`

6.2.6 Neuronová síť MLP pomocí knihovny `Keras`

Knihovna `Keras` nabízí mnohem více možností při definování neuronové sítě MLP než knihovna `scikit-learn`. Zejména umožňuje definovat jednotlivé vrstvy neuronové sítě různých typů, včetně nastavení aktivační funkce pro jednotlivé vrstvy zvlášť. V rámci experimentů byly použity dvě skryté vrstvy perceptronů s aktivační funkcí *ReLU*, mezi nimiž je ještě tzv. *dropout* vrstva. Princip této vrstvy spočívá zejména v náhodném zahazování trénovacích vzorků, přičemž při predikci už tato vrstva samozřejmě použita není. To ve svém důsledku vede ke snížení pravděpodobnosti, že natrénovaná síť bude trpět problémem přeučení [15]. Výstupní vrstva pak obsahuje jeden perceptron s sigmoidální aktivační funkcí, na základě které je rozhodnuto o výsledné třídě. Jako optimalizační algoritmus při trénování byl použit algoritmus Adam [7].

Velkou výhodou dále je, že klasifikátoru vytvořenému pomocí knihovny `Keras` lze předat parametrem váhy cílových tříd na základě jejich zastoupení, čímž je vyřešen problém nevyváženosti dat. Pro další snížení možnosti přetrénování neuronové sítě bylo ještě využito principu tzv. *early stopping*, což znamená, že učení je ukončeno dříve v případě, že se chyba na trénovacích datech stále snižuje, nicméně chyba na datech testovacích se již začne zvyšovat, respektive nesníží se po určitý počet kroků. Tím je zabráněno tomu, že by se klasifikátor příliš specializoval na trénovací množinu a následně nebyl schopný generalizace.

Na obrázku 6.6 je vidět výsledná matice záměn. Dosažená míra *true positive* je velmi slušná, sice za cenu zvýšení míry *false positive*, nicméně nejedná se o tak drastické zvýšení jako v případě metody SVM (viz obrázek 6.4). Míra *false negative* je relativně nízká, takže dosažené výsledky jsou ve srovnání s ostatními metodami velmi dobré.



Obrázek 6.6: Matice záměn pro metodu MLP pomocí knihovny `Keras`

Výsledné hodnoty vybraných metrik je pak možno nalézt v tabulce 6.6. Hodnota metriky *recall* u minoritní třídy je velmi dobrá, na druhou stranu se ve shodě s očekáváním snížila hodnota *precision* – tomu pak odpovídá i snížení hodnoty jejich harmonického průměru, tedy f_1 *score*. Oproti tomu metrika AUC-ROC vykazuje nejlepší hodnotu z doposud zkoumaných – hodnotu 0,99. Z hlediska praktického využití se zejména díky vynikající míře *recall* tento klasifikátor jeví jako nejlepší.

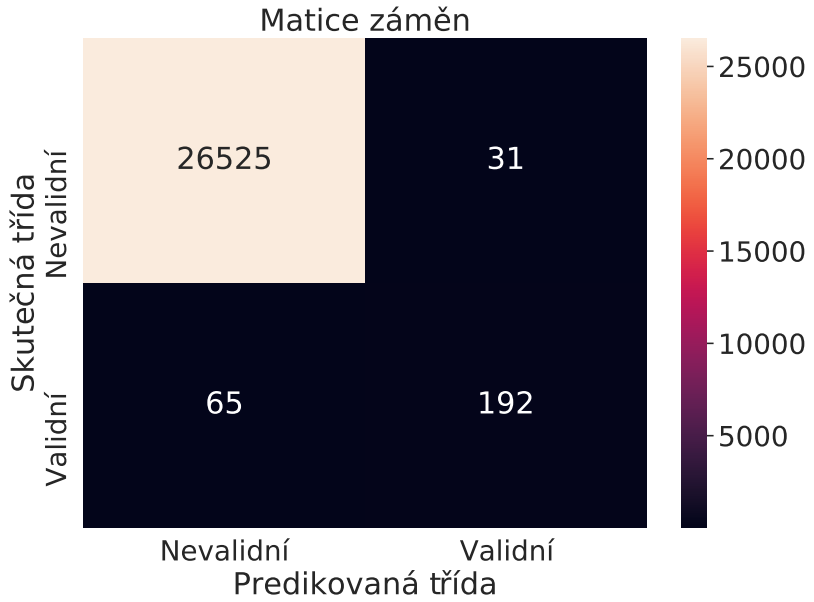
	support	precision	recall	f1-score
0	26556	1,00	0,98	0,99
1	257	0,35	0,91	0,50
macro avg		0,67	0,95	0,75
weighted avg		0,99	0,98	0,99
accuracy			0,98	
AUC-ROC			0,99	

Tabulka 6.6: Výsledky metrik pro klasifikátor MLP z knihovny `Keras`

6.2.7 Soubor více klasifikátorů s hlasováním

Dále byly provedeny experimenty se souborem klasifikátorů, kdy výsledek je vybrán na základě hlasování. Implementaci této metody poskytuje knihovna `scikit-learn` v podobě třídy `VotingClassifier`. Pro hlasování byly použity klasifikátory náhodný les, KNN, SVM a MLP se stejným nastavením parametrů jak u výše popsaných experimentů.

Výsledná predikce na testovacích datech v podobě matice záměn je pak zobrazena na obrázku 6.7. Jde vidět, že výsledky se nijak dramaticky neliší od výsledků dílčích použitých klasifikátorů, což značí to, že chybně hodnocené vzorky testovacích dat se napříč klasifikátory víceméně shodují.



Obrázek 6.7: Matice záměn pro metodu hlasování

Tabulka 6.7 pak ukazuje hodnoty jednotlivých vybraných metrik. Hodnoty se příliš neliší od dílčích použitých klasifikátorů, což odpovídá matici záměn. Výjimku představuje metrika AUC-ROC, která v tomto případě nemohla být spočítána, neboť vzhledem k tomu, že výsledná třída je určena hlasováním na základě výsledků jednotlivých klasifikátorů, nemá tento soubor klasifikátorů jako celek možnost určit skóre, tedy pravděpodobnost příslušnosti k pozitivní třídě, jednotlivých testovacích vzorků. Proto hodnota metriky AUC-ROC v tomto případě chybí.

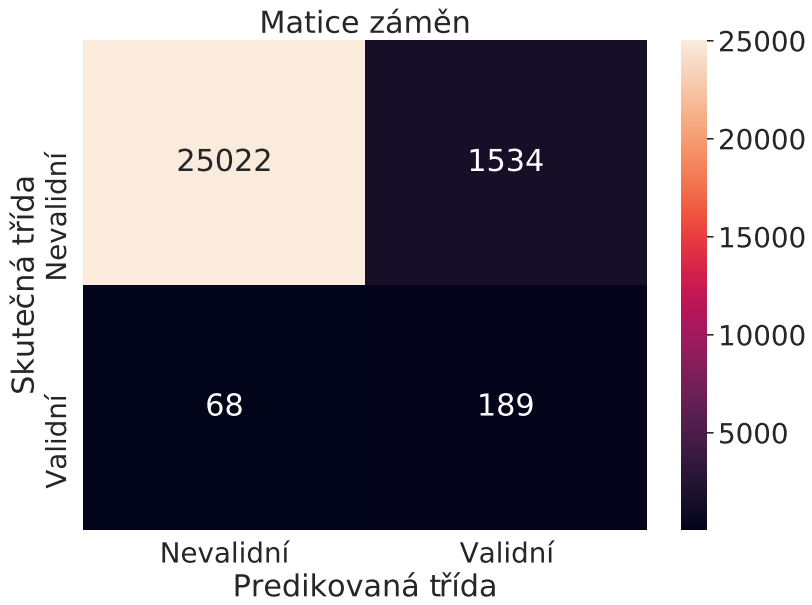
	support	precision	recall	f1-score
0	26556	1,00	1,00	1,00
1	257	0,86	0,75	0,80
macro avg		0,93	0,87	0,90
weighted avg		1,00	1,00	1,00
accuracy			1,00	
AUC-ROC			-	

Tabulka 6.7: Výsledky metrik pro hlasování

6.2.8 AdaBoost

Dále byly provedeny experimenty, zda by se nepodařilo zvýšit úspěšnost metody SVM pomocí algoritmu AdaBoost. Tento algoritmus implementuje třída `AdaBoostClassifier` z knihovny `scikit-learn`. Princip tohoto algoritmu spočívá ve váženém hlasování souboru klasifikátoru, přičemž váhy jednotlivým klasifikátorům se nastavují na základě jejich chybovosti.[3] Dílčími klasifikátory jsou v tomto případě klasifikátory SVM, se stejnými parametry jak bylo popsáno výše, tedy s lineární kernelovou funkcí a regularizačním parametrem $C = 10$.

Obrázek 6.8 ukazuje výsledky na testovací množině v podobně matice záměn. Je vidět, že v porovnání s maticí záměn pro metodu SVM 6.4 došlo k mírnému snížení míry *false positive* nicméně se zvýšila míra *false negative*.



Obrázek 6.8: Matice záměn pro algoritmus AdaBoost

Tabulka 6.8 zachycuje výsledky vybraných metrik pro algoritmus AdaBoost nad klasifikátory SVM. Dle očekávání je velmi špatná hodnota *precision*, dokonce horší než u metody SVM 6.4, což je dáno poklesem míry *true positive*. Hodnota metriky *recall* se v porovnání s experimenty s jedním klasifikátorem SVM rovněž zhoršila, celkově se tedy dá říct, že z hlediska praktického využití se jedná o dokonce horší výsledky, než v případě jednoho klasifikátoru SVM.

	support	precision	recall	f1-score
0	26556	1,00	0,94	0,97
1	257	0,11	0,74	0,19
macro avg		0,55	0,84	0,58
weighted avg		0,99	0,94	0,96
accuracy			0,94	
AUC-ROC			0,89	

Tabulka 6.8: Výsledky metrik pro AdaBoost

6.3 Shrnutí a výběr modelu

Z výše provedených experimentů a výpočtů metrik pro testovací množinu dat vyplývá, že zatímco majoritní třídu se prakticky všem klasifikátorům, s nimiž bylo experimentováno, daří velmi dobře klasifikovat, u minoritní třídy se výsledky velmi různí. Z praktického hlediska vychází jako nejlepší kandidáti dvě metody – neuronová síť MLP implementovaná pomocí knihovny `Keras` (viz obrázek 6.6), klasifikátor Náhodný les a soubor některých klasických klasifikátorů s výběrem na základě majoritního hlasování (viz obrázek 6.7).

Neuronová síť MLP dosahuje zejména velmi dobrých výsledků z hlediska metrik *recall* a `AUC-ROC`. V praxi systém bude používán tak, že nabídne uživateli pro danou platbu několik nejpravděpodobnějších faktur, ke kterým by se tato platba mohla vztahovat. Vzhledem k výsledkům uvedeným v matici záměn a na základě metriky *precision*, které je pro pozitivní třídu rovna hodnotě $0,35$, by tak v tomto případě na jednu správnou fakturu v nabídce (*true positive*) připadaly dvě nesprávně přiřazené faktury (*false positive*). Zároveň je velmi malá pravděpodobnost, že se některá z faktur, za niž byla daná platba uskutečněna, v nabídce neobjeví, neboť metrika *recall* pro pozitivní třídu dosahuje hodnoty $0,91$. Z praktického úhlu pohledu se tak tento model jeví jako nejvhodnější.

Oproti tomu klasifikátory typu náhodný les či hlasování sice dosahují výborných výsledků co do metriky *precision* u minoritní třídy, nicméně metrika *recall* pro minoritní třídu v těchto případech nedosahuje ani hodnoty $0,8$, což by způsobilo, že relativně velká část faktur, jež reálně spadají do pozitivní třídy, by se do nabídky pravděpodobných faktur ani nedostala, což je z hlediska praktické využitelnosti problém.

S ohledem na výše uvedené skutečnosti tedy byl pro implementaci predikce a integrace do firemního systému nakonec zvolen klasifikátor neuronové sítě MLP implementovaný pomocí knihovny `Keras`.

Kapitola 7

Závěr

Cílem této diplomové práce bylo navrhnout způsob identifikace plateb zejména v případě, že chybí variabilní symbol, jak je tomu typicky u zahraničních plateb. Pro řešení tohoto problému byl zvolen data mining, respektive strojové učení. Zadavatelem mé diplomové práce je firma Platební instituce Roger a.s. V této kapitole bude následně provedeno zhodnocení dosažených výsledků a návrh možných rozšíření systému do budoucna.

7.1 Zhodnocení

Po úvodním seznámení se s problematikou a dodanými daty bylo nejprve nutné navrhnout způsob řešení. Z hlediska získávání znalostí se nejedná o úplně typickou úlohu, neboť princip identifikace plateb spočívá v nalezení vazeb mezi aktuálně platnými fakturami a příchozími platbami, tedy de facto doplnění identifikátorů plateb a transakcí do vazební tabulky. Predikovat identifikátory ve vazební tabulce ovšem principiálně není možné, proto bylo nutné tento problém uchopit jinak a navrhnout jiný způsob predikce tak, aby byl principiálně proveditelný. Jako řešení tohoto problému byl navržen a implementován algoritmus, jehož princip by se dal zjednodušeně shrnout následujícím způsobem: nejprve jsou vytvořeny všechny potenciální kombinace faktur a transakcí a následně je provedena klasifikace do dvou tříd na základě toho, zda daná kombinace je či není správná. Tímto způsobem je identifikace plateb převedena na klasický problém řešený pomocí strojového učení – binární klasifikaci.

V další fázi byl tento navržený postup implementován, přičemž pro řešení bylo využito několika metod, jež lze na binární klasifikaci aplikovat. Následně byly provedeny experimenty na testovacích datech, které vyhodnocovaly schopnost těchto metod správně predikovat validitu kombinací faktur a plateb pomocí více různých metrik, přičemž bylo zvažováno i praktické hledisko, tedy zejména z hlediska pravděpodobnosti výskytu správných faktur v množině faktur predikovaných implementovaným nástrojem. Jako nejlepší z těchto experimentů vyšel klasifikátor založený na neuronové síti perceptronů, jež byla implementována s pomocí knihovny `Keras`. Tento klasifikátor byl tedy následně využit při integraci nástroje do interního systému firmy Platební instituce Roger a.s., kde slouží pro predikci pravděpodobných faktur k dané platbě v případě chybějícího variabilního symbolu.

Kromě identifikace platby s chybějícím variabilním symbolem samozřejmě implementovaný nástroj dokáže platbu identifikovat i na základě variabilního symbolu, nicméně tato funkcionalita je vcelku triviální a nevyžaduje použití strojového učení, takže není předmětem mé diplomové práce.

Systém bude s drobnými úpravami využitelný i jako služba pro třetí strany. Jedna z věcí, která však bude velmi pravděpodobně vyžadována, je získání dat pro trénování modelu od konkrétního subjektu, který by chtěl systém využívat a následně klasifikátor inkrementálně přetrénovat. Bez provedení tohoto kroku je totiž relativně vysoká šance, že by se přesnost predikce snížila. Avšak z hlediska implementace nic zásadního nebrání využití tohoto nástroje třetími stranami.

7.2 Možnosti pokračování

Z hlediska budoucího vývoje existuje několik možností, na které bude přínosné se zaměřit. Za dvě nejvýznamnější oblasti z hlediska pokračování tohoto projektu bych označil zejména způsoby pro zpřesnění výsledků predikce a automatizaci trénování modelu. Tyto možnosti rozšíření tedy budou dále rozebrány podrobněji.

7.2.1 Automatizace trénování modelu

První plánem do budoucna a věcí, které bude dobré věnovat úsilí, je automatické trénování modelu a sběr dat. Je totiž více než pravděpodobné, že s přibývajícím časem bude klesat i přesnost predikce. Z toho důvodu bude nezbytné implementovat funkcionalitu, která po určité době provede inkrementální přetrénování modelu na základě aktuálních dat. Jednak tedy bude nutné data automaticky sbírat a následně pak automaticky spouštět inkrementální trénování. Použitý model neuronové sítě MLP z knihovny `Keras` inkrementální trénování podporuje, takže implementace této funkcionality pravděpodobně nebude představovat zásadní problém.

7.2.2 Zpřesnění predikce

Asi největší nevýhodou využitého klasifikátoru, tedy neuronové sítě perceptronů pomocí knihovny `Keras`, byla poměrně vysoká míra tzv. *false positive* vzorků, tedy kombinací faktur a transakcí, jež jsou nástrojem označeny jako pravděpodobné, avšak které k sobě ve skutečnosti nepatří. V první fázi bude tedy budoucí vývoj nejspíše zaměřen tímto směrem. Z tohoto důvodu zde nastíním některé možnosti, které by mohly zpřesnění poskytnout a jež by tudíž mohlo být dobré prozkoumat. Samozřejmě platí, že se jedná spíše o návrhy, takže nelze zaručit, že skutečně k zpřesnění výsledků povedou.

Kombinace klasifikátorů

Jedním z možných způsobů, který by mohl vést k přesnějším výsledkům je kombinace klasifikátorů. Tím ovšem není myšleno přímo sdružování s hlasováním, nýbrž spíše řetězení – nejprve by byla provedena predikce pomocí modelu neuronové sítě perceptronů a teprve následně by byla provedena další zpřesňující predikce dalším klasifikátorem. Vzhledem k tomu, že množina potenciálních kombinací, jež je vrácena implementovaným prediktorem, je mnohem vyváženější než původní data, mohlo by to vést k dodatečnému zpřesnění. Nicméně tato navržená metoda je spíše spekulativní a skutečný dopad na přesnost bude samozřejmě první nutné ověřit experimenty.

Subset sum problem

Další možností, jak výsledky zpřesnit, je srovnání nominálních hodnot faktur a zaplacených částek. Jak již bylo řečeno, obecně platí, že vazba mezi fakturami a transakcemi je typu M:N, neboť jedna platba může znamenat úhradu více faktur, ale i pouze části faktury. Z toho důvodu je nutné hledat skupinu faktur, u kterých se součet nominálních hodnot rovná součtu částek určité skupiny transakcí. Typově se ovšem jedná o tzv. *subset sum problem*, respektive se na něj dá snadno převést například tak, že nominální hodnota faktur bude brána jako záporná, částka transakce jako kladná, načež cílem bude nalézt podmnožinu, jejíž součet je roven nule. Tento problém představuje známý problém z kategorie NP-úplných problémů, tedy problémů, jež nejsou deterministicky řešitelné v polynomiálním čase. Bohužel v tomto případě ani není možné pro řešení využít některých optimalizačních algoritmů, které se pro řešení těchto typů úloh běžně používají, neboť v praxi je nutné, aby se částky mezi sebou rovnaly přesně, což znamená řešení hrubou silou. Proto tento způsob není pro identifikaci plateb příliš využitelný sám o sobě, nicméně by mohl být využit ve spojení s implementovaným prediktorem, neboť by mohl posloužit k dodatečnému vyfiltrování potenciálních faktur po provedení predikce v případě, že by se podařilo odpovídající součet nalézt. Jelikož by se v tomto případě jednalo o podstatně menší množinu, bylo by teoreticky možné i řešení tohoto problému hrubou silou.

Vzhledem k tomu, že v případě úhrady pouze části faktury však další platby mohou dorazit až později, se nemusí vždy podařit takový součet nalézt. Z toho důvodu není možné tuto metodu použít vždy, nicméně pokud se takový součet v predikované množině podaří nalézt, je pravděpodobnost, že se jedná o správné kombinace faktur a transakcí, velmi vysoká.

Literatura

[1] Boschetti, A. a Massaron, L. *Python Data Science Essentials*. 2. vyd. Birmingham, UK: Packt Publishing, 2016. ISBN 978-1-78646-213-8.

[2] *Classification on imbalanced data* [online]. 2020 [cit. 2020-05-14]. Dostupné z: https://www.tensorflow.org/tutorials/structured_data/imbalanced_data.

[3] Han, J., Kamber, M. a Pei, J. *Data Mining: Concepts and Techniques*. 3. vyd. Elsevier Science, 2011. The Morgan Kaufmann Series in Data Management Systems. ISBN 0123814790.

[4] Idris, I. *Python Data Analysis Cookbook*. 1. vyd. Birmingham, UK: Packt Publishing, 2016. ISBN 978-1-78528-228-7.

[5] Jeni, L., Cohn, J. a De la Torre, F. Facing Imbalanced Data - Recommendations for the Use of Performance Metrics. In:. Září 2013, sv. 2013. DOI: 10.1109/ACII.2013.47.

[6] *Keras API reference* [online]. 2020 [cit. 2020-05-14]. Dostupné z: https://keras.io/api/.

[7] Kingma, D. P. a Ba, J. *Adam: A Method for Stochastic Optimization*. 2014 [cit. 2020-05-14]. Dostupné z: https://arxiv.org/abs/1412.6980.

[8] Layton, R. *Learning Data Mining with Python*. 1. vyd. Birmingham, UK: Packt Publishing, 2015. ISBN 978-1-78439-605-3.

[9] *NumPy Reference* [online]. 2020 [cit. 2020-05-14]. Dostupné z: https://numpy.org/devdocs/reference/index.html.

[10] McKinney, W. et al. *Pandas documentation* [online]. 2020 [cit. 2020-05-14]. Dostupné z: https://pandas.pydata.org/docs/index.html.

[11] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*. 2011, sv. 12, s. 2825–2830.

[12] Raschka, S. *Python Machine Learning*. 1. vyd. Birmingham, UK: Packt Publishing, 2016. ISBN 978-1-78355-513-0.

[13] *Scikit-learn guide* [online]. 2020 [cit. 2020-05-14]. Dostupné z: https://scikit-learn.org/stable/user_guide.html.

[14] *SciPy Reference* [online]. 2020 [cit. 2020-05-14]. Dostupné z: https://docs.scipy.org/doc/scipy/reference.

[15] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. a Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 2014, sv. 15, č. 56, s. 1929–1958. Dostupné z: http://jmlr.org/papers/v15/srivastava14a.html.

[16] *TensorFlow* [online]. 2020 [cit. 2020-05-14]. Dostupné z: https://www.tensorflow.org.

Příloha A

Obsah přiloženého paměťového média

- `src` – zdrojové soubory implementovaného systému pro identifikaci plateb
- `doc` – zdrojové soubory technické zprávy diplomové práce v systému LaTeX
- `README.md` – návod k instalaci a spuštění
- přeložená technická zpráva ve formátu PDF