Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Predikce zpoždění letů metodou strojového učení

Bakalářská práce

Autor: Zdeněk Archleb
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. RNDr. Kamila Štekerová, Ph.D., MSc.

Hradec Králové Srpen 2023

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 14.8.2023 Zdeněk Archleb

Poděkování:

Děkuji vedoucí mé bakalářské práce doc. RNDr. Kamile Štekerové, Ph.D., MSc. za pomoc s výběrem tématu, mnoho užitečných rad a doporučení v pro mě nové oblasti informačních technologií a provedení celým tímto procesem tvorby BP.

Anotace

Hlavním cílem této teze je popsání metody rozhodovacích stromů v rámci strojového učení a její aplikace pro predikci zpoždění letů. K tomu je využito algoritmu CART rozšířeného o gradient boosting za pomoci nové Python knihovny CatBoost, a dalších běžně využívaných knihoven Scikit, Pandas a matplotlib.

Annotation

Title: Prediction of flight delays using machine learning

The main goal of this thesis is to describe the decision tree method within machine learning and its application for predicting flight delays. To this end, the CART algorithm is utilized, enhanced with gradient boosting with the help of the new Python library CatBoost, along with other commonly used libraries such as Scikit, Pandas, and matplotlib.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Zpoždění letů může mít dalekosáhlé ekonomické dopady. Ze schopnosti předpovědět zpoždění letů mohou mít prospěch všichni účastníci tohoto procesu. Zvýší se spokojenost cestujících, kteří si mohou lépe rozvrhnout svůj čas, řízení letiště bude schopno lépe plánovat své logistické operace a letecké společnosti mohou optimalizovat svůj provoz. Tím však nejdůležitějším faktorem jsou peníze. Zpoždění letů stojí každý rok miliardy amerických dolarů, a tak každá činnost, která může přispět k jejich snížení je velmi důležitá. (Ball et al., 2010)

Cílem této práce je popsat problematiku zpoždění letů a vytvořit predikční model za využití dostupných datasetů. Na základě článku (Tang, 2021) byl pro předpověď zpoždění letů zvolen algoritmus rozhodovacích stromů, který byl v průběhu této práce rozšířen o gradient boosting z Python knihovny CatBoost.

2 Teoretická část

2.1 Úvod do problematiky zpoždění letů

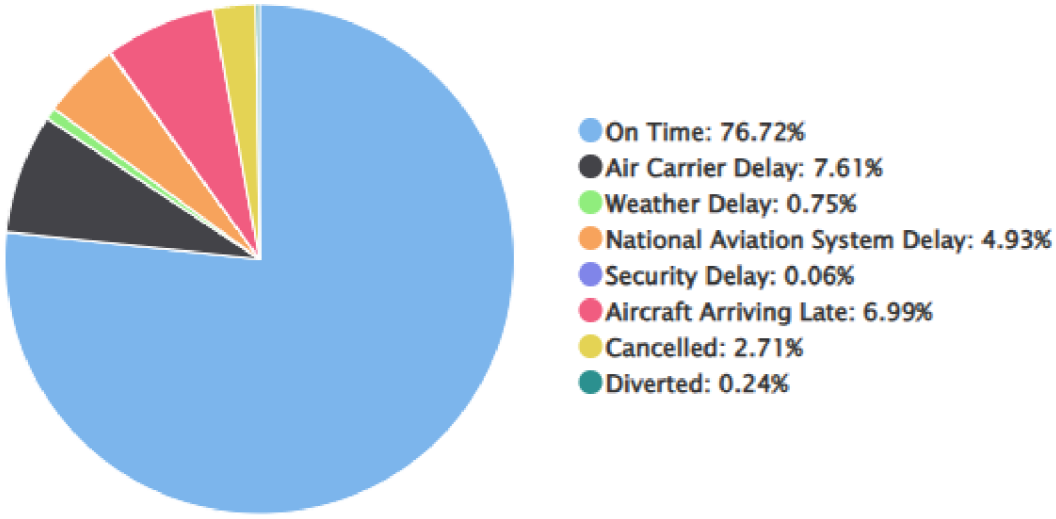
2.1.1 Finanční dopady zpoždění letů

Během minulého století letecké společnosti vyrostly z obyčejných poštovních dopravců do intelektuálně zajímavých společností. Letecký průmysl je velmi kompetetivní, dynamický a má v sobě hodně náhodných parametrů, z čehož poté vznikají nejistoty. Jednou z těchto nejistot je zpoždění letu, ke kterému přispívá velké množství faktorů, jako je špatné počasí, fyzikální zákony, zpožděné přílety a problémy spojené s posádkou. (Khaksar, A. 2017)

Podle studie (Ball, M., Barnhart, C., and Drenser, M., 2010) je zpoždění v leteckém průmyslu vážný a rozsáhlý problém ve Spojených státech. Stále se zvyšující letová zpoždění významně zatěžují cestovní systém Spojených států a stojí aerolinky, cestující a lidskou společnost mnoho miliard dolarů každý rok.
Podle odhadu týmu, který stojí za Total Delay Impact Study (Ball et al., 2010) byly celkové náklady spojené se zpožděními v letecké dopravě Spojených států v roce 2007 32,9 miliard amerických dolarů.

2.1.2 Příčiny zpoždění letů

Za zpožděný je podle amerického úřadu pro transportační statistiky (BTS, n.d.) považován let, který přiletěl do cílové destinace o 15 minut nebo déle než bylo plánováno.



Obrázek 1: Příčiny zpoždění letů v roce 2022
Zdroj: (BTS.gov, n.d.).

Podle dat BTS (Bureau of Transportation Statistics) za celý rok 2022 přiletělo na čas 76,72% všech letů, respektive se zpožděním menším než 15 minut.

Nejčastější příčina zpoždění letů byla Air Carrier Delay (7,61%), což podle BTS znamená zpoždění, nad kterým dopravce nemá kontrolu. Řadí se mezi ně: čištění letadla, poškození letadla, čekání na navazující spoj cestujících nebo posádky, naložení zavazadel a jiného nákladu, kolize s ptactvem, catering – zpoždění spojené s obstaráním jídla, problémy s výpočetními zařízeními, výpadek proudu, zákonem nařízený odpočinek posádky, poškození nebezpečného nákladu, technická inspekce, doplňování paliva, péče o handicapované pasažéry, pozdní příchod posádky, servis záchodů, údržba, přeprodeje - prodání více lístků než je míst v letadle, servis pitné vody, vyvedení neukázněných cestujících, pomalé nastoupení nebo usazení pasažérů, ukládání příručních zavazadel, váhová a rovnováhová zpoždění - zpoždění, kdy se z důvodu bezpečnosti musí rovnoměrně rozmístit náklad nevyváženého letadla z důvodu bezpečnosti. Samozřejmě může nastat i několik takovýchto nečekaných komplikací současně.

Na druhém místě je pak pozdní přílet letadla (6,99%). Hlavní problém tohoto typu zpoždění je ten, že tím může vzniknout dominový efekt, kdy i na každé další letiště toto letadlo přiletí pozdě. Tomuto dominovému efektu se v angličtině podle BTS říká „delay propagation“. Zpoždění může ovlivnit nejen navazující lety,

ale i lety, které byly plánovány na stejný letištní terminál. Když letadlo přiletí pozdě, a nestihne svůj plánovaný odlet, musí dostat nový termín odletu. Změna se dotkne i dalších odletů ze stejného terminálu nebo i letiště.

Na třetím místě je NAS (National Airspace System, v češtině Národní systém vzdušného prostoru) delays, což je zpoždění, které je pod kontrolou NAS (4,93%). Tento druh zpoždění může být ovlivněn počasím, velkým množstvím letecké dopravy, řízením letového provozu. Zpoždění, která nastanou po otevření brány jsou obvykle připisována NAS.

Dalšími příčinami zpoždění letů jsou zrušené nebo odkloněné lety (2,71%), extrémní počasí (0,75%), zpoždění z důvodu bezpečnosti (0,06%).

2.1.3 Závěry vyplývající z množství příčin zpoždění letů

Podle dat BTS z roku 2022 je letecké odvětví výrazně ovlivněno mnoha různými faktory, které mohou způsobit zpoždění letů. Nejčastější příčiny zpoždění letů jsou Air Carrier Delay - pozdní přílet letadla a NAS Delay, které společně tvoří většinu zpoždění. Je zřejmé, že i malé detaily, jako například údržba letadla nebo pomalé nastupování pasažérů mohou mít významný vliv na zpoždění letů.

Na základě výše uvedených informací je patrné, že na zpoždění v letecké dopravě má vliv velké množství faktorů. Letecké společnosti shromažďují o letech mnoho dat, na základě nichž lze tvořit modely strojového učení a detekovat v datech vzory, které by běžný člověk těžko nalezl. To umožní řídícím střediskům včas detekovat let, který může být potenciálně zpožděn a připravit taková opatření, která zamezí negativním jevům jako je delay propagation.

2.2 Strojové učení

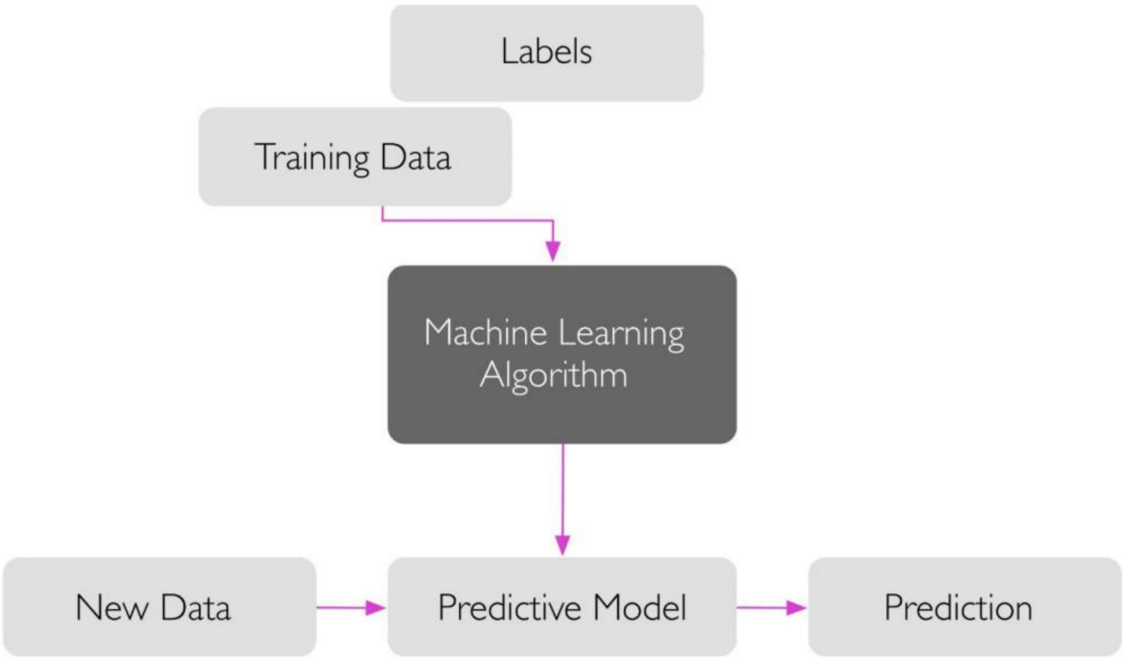
Strojové učení je odvětví umělé inteligence, které se zaměřuje na vývoj algoritmů, jež umožňují počítačům učit se z dat a provádět predikce nebo rozhodování na jejich základě. Tyto algoritmy mohou být trénovány na základě učení se s učitelem, kdy algoritmus dostane data, která obsahují výstupní hodnotu,

učení se bez učitele, kdy vstupní data nemají přiřazenou výstupní hodnotu a učení posilováním, kdy se algoritmus za pomoci interakcí snaží maximalizovat nějakou odměnu. Tato práce je zaměřena na učení s učitelem.

2.2.1 Učení s učitelem

Hlavním cílem učení s učitelem (supervised learning) je vytrénovat model z klasifikovaných trénovacích dat tak, aby nám umožnil dělat predikce na datech, která v současnosti nemá, a budou mu v budoucnosti předložena. Učení s učitelem se využívá pro úlohy klasifikace a regrese. (Sebastian RASCHKA a Vahid MIRJALILI, 2019).

Na obrázku (Obr. 2) je znázorněn model strojového učení s učitelem. Máme zde trénovací data označená popiskem (označením třídy, tzv. labelem). Data jsou na vstupu algoritmu strojového učení, který trénovací data zpracuje. Výstupem zpracování dat je prediktivní model. Do modelu vstupují nová data, pro která prediktivní model vypočte (předpoví) jejich popis (třídu, label).



Obrázek 2: Učení s učitelem
Zdroj: (RASCHKA, Sebastian a Vahid MIRJALILI, 2019).

2.2.2 Klasifikace a regrese

V modelech strojového učení se pracuje s proměnnými, které jsou charakterizovány buď jako kvantitativní (číselné, numerické) nebo kvalitativní (kategorické). (James et al., 2021)

Příklady kvantitativních proměnných jsou časová délka zpoždění, čas odletu nebo množství paliva v nádrži. Příklady kvalitativních proměnných jsou název letiště a typ přepravce.

Někdy máme tendenci odkazovat na kvantitativní úlohy jako na regresivní, zatímco ty, které mají kvalitativní ráz často označujeme za klasifikační. Rozdíl však není vždy tak jasný. Lineární regrese metodou nejmenších čtverců se používá s kvantitativními parametry, zatímco logistická regrese se typicky používá s kvalitativními parametry, ale protože odhaduje pravděpodobnost příslušnosti ke třídě, může se o ní také hovořit i jako o regresivní metodě. Některé metody jako např. metoda K-nejbližších sousedů (K-nearest neighbours), mohou být použity v kvantitativních i kvalitativních úlohách. (James et al., 2021) V případě algoritmu CART, kterému je v této práci dále věnována pozornost, je možné jeho využití jak pro klasifikační tak pro regresivní úlohy.

Klasifikace:

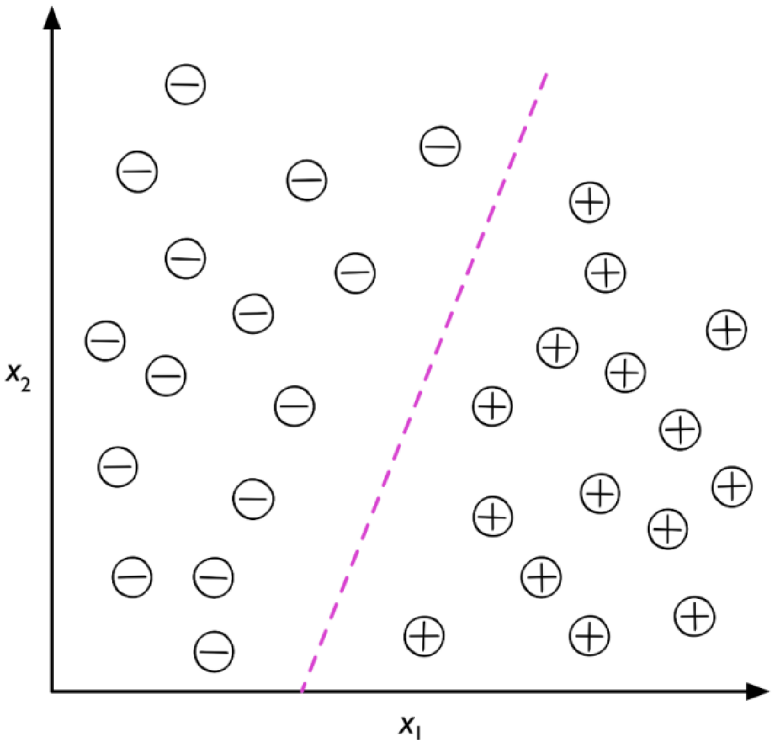
Jedná se o podkategorii učení se s učitelem, kdy se algoritmus učí z datasetu s označenými daty, tedy daty, u kterých víme, co reprezentují, a snažíme se tato označení přiřadit dalším neoznačeným datům. Tato označená data jsou diskrétní, neuspořádané hodnoty, které mohou být chápány jako skupiny členství v instancích. (Reschka, 2019)

Z hlediska této práce uvažujeme binární klasifikaci, tedy zařazování do dvou tříd. Jev buď nastal nebo nenastal, resp. let byl zpožděn nebo nebyl zpožděn.

Dalším příkladem, který je velmi často uváděn v rámci různých článků nebo knih (Rachel, 2020) (Reschka, 2019) je využití klasifikace pro filtrování e-mailů, kdy se ze vzorku e-mailů označených jako spam nebo non-spam na základě jejich

parametrů vytvoří model, který bude umět vyhodnotit, zda další příchozí e-maily zařadit do kategorie spam, či nikoliv.

Na obrázku (Obr.3) je uveden příklad binární klasifikace, kde se algoritmus učí na základě 30 trénovacích vzorků. Vzorky jsou rozděleny na 15 kladných a 15 záporných. Každý vzorek je tvořen hodnotami x_1 a x_2. Algoritmus se snaží najít nějaké pravidlo, které by oddělilo kladné vzorky od záporných. Rozdělení je reprezentováno přerušovanou čarou.



Obrázek 3: Příklad binární klasifikace, kde se algoritmus učí rozdělovat kladná a záporná data na základě parametrů x_1 a x_2
Zdroj: (RASCHKA, Sebastian a Vahid MIRJALILI, 2019).

Klasifikace však nemusí obsahovat pouze binární labeled class data, ale může obsahovat jakýkoliv počet tříd. Například dataset pro rozpoznávání ovoce by mohl obsahovat labeled classes [„Jablka“, „Třešně“, „Hrušky“].

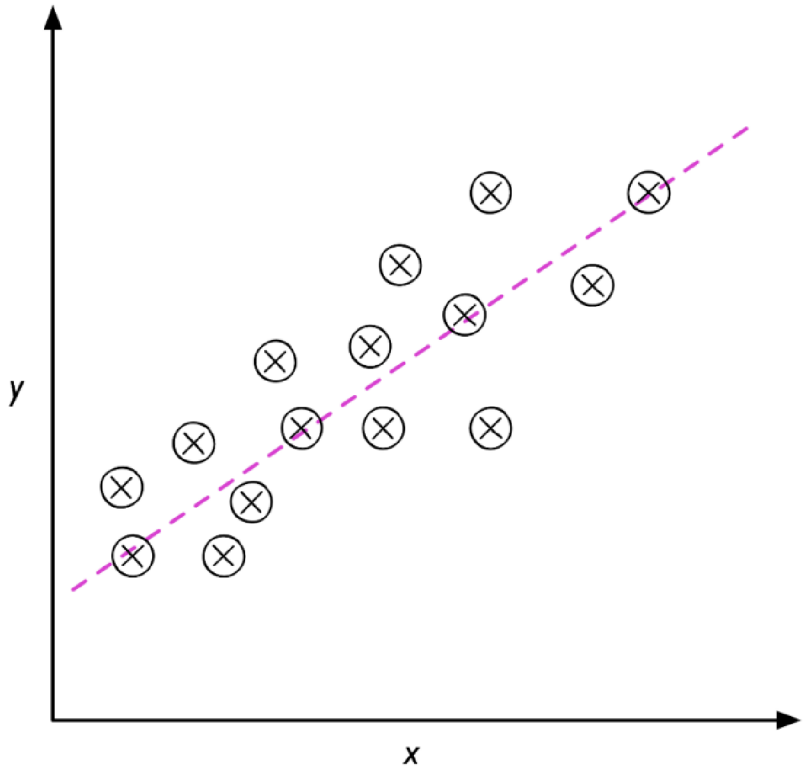
Regrese:

V regresivní analýze dostáváme jeden nebo více vstupních atributů pro předpověď spojité (kontinuální) hodnoty. Stejně jako v klasifikaci je cílem najít vztah mezi vstupními proměnnými a výstupní proměnnou tak, aby byl schopen co

nejlépe předpovědět výstupní hodnoty pro nová, dosud neznámá data. (Raschka, 2019)

Příkladem regrese by mohlo být například pokud bychom chtěli předpovědět, jak dlouhé zpoždění může u nějakého konkrétního letu nastat.

Na obrázku 4 je příklad lineární regrese, kde je 15 bodů tvořených hodnotami x a y. Algoritmus lineární regrese, který je znázorněn přerušovanou čarou se snaží najít přímku, která co nejlépe odpovídá bodům.



Obrázek 4: Příklad lineární regrese, kde se snažíme minimalizovat vzdálenost úsečky k datům
Zdroj: Zdroj: (RASCHKA, Sebastian a Vahid MIRJALILI, 2019).

2.3 Rozhodovací stromy

2.3.1 Výhody a nevýhody rozhodovacích stromů

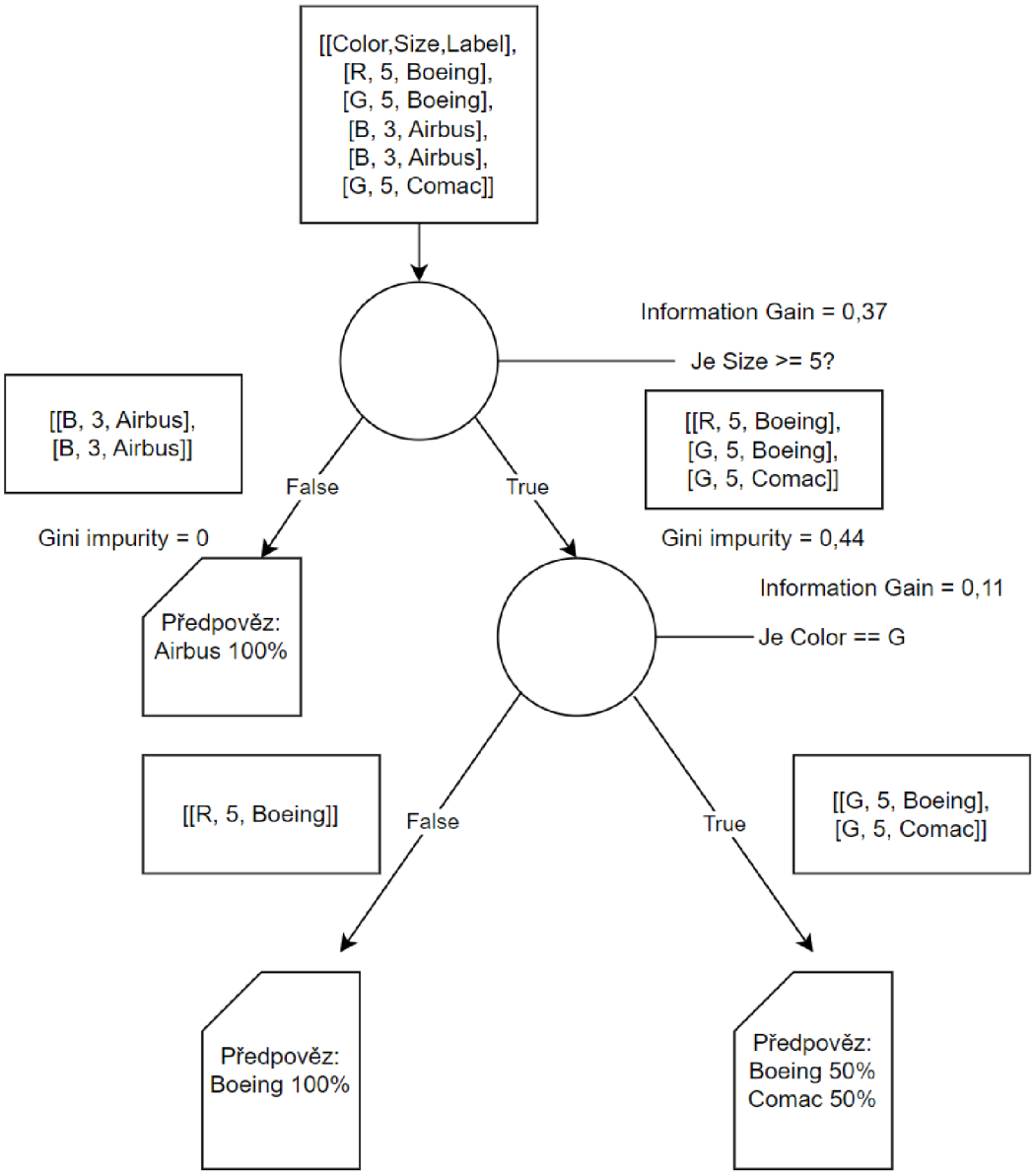
Podle (James et al., 2021) je jednou z výhod to, že rozhodovací stromy jsou jednoduché na vysvětlení lidem, protože mají blíže k tomu, jak se lidé rozhodují (v porovnání s regresí). Pokud se zamyslíme nad tím, jakým způsobem se

rozhodujeme ve svém každodenním životě, můžeme najít podobnosti s rozhodovacími stromy. Dalšími výhodami, které (James et al., 2021) zmiňuje, jsou jednoduchá grafická vizualizace, což se váže k předchozím bodům a jednoduché zvládání kvalitativních prediktorů.

Jako nevýhody rozhodovacích stromů (James et al., 2021) uvádí to, že stromy obecně nemají stejnou úroveň prediktivní přesnosti, což se ovšem dá řešit například pomocí boostingu nebo spojení více stromů do náhodného lesa (Reschka, 2019). Jako další nevýhodu (Reschka, 2019) uvádí, že rozhodovací stromy jsou velmi nerobustní. To znamená, že i malá změna v datech může způsobit velkou změnu v rozhodovacím stromu.

2.3.2 Obecná struktura CART pro klasifikaci

Na obrázku (Obr. 5) je model rozhodovacího stromu pro klasifikační úlohu rozpoznání typu letadel na základě dvou jeho parametrů – barvy a velikosti. V této kapitole bude z tohoto modelu stromu vycházeno pro vysvětlení jeho fungování. Obdélníky slouží k reprezentaci datasetu, který vstupuje do jednotlivých uzlů. Jednotlivé uzly jsou reprezentovány kruhy. Spojení, neboli větve, jsou reprezentovány pomocí šipek, které vycházejí z jednotlivých uzlů na základě odpovědi na otázku, kde odpověď je buď true nebo false. U uzlů je znázorněn information gain a u datasetů gini impurity. Jednotlivé listy tohoto rozhodovacího stromu jsou reprezentovány zkoseným obdélníkem.



Obrázek 5: Model rozhodovacího stromu pro klasifikaci, vlastní tvorba. Inspirováno: (Gordon, 2017), vlastní zpracování.

Kořenový uzel:

Root node, neboli kořenový uzel, je uzel, který je na začátku vzniku rozhodovacího stromu. Do kořenového uzlu vstupuje celý dataset, který se na základě podmínky, která mu přinese nejvyšší gini index, rozdělí na dva menší datasety.

Uzly:

Uzel funguje na stejném principu jako kořenový uzel. Dostává do sebe dataset z předchozího uzlu podle jeho rozdělovací otázky a snaží se dataset dále rozdělit podle nejvyššího gini indexu. Pokud gini index vyjde nulový, stává se z uzlu list.

Rozdělovací otázka:

Tato otázka určuje, podle jakého parametru a jaké hodnoty se dále rozdělí rozhodovací strom, popřípadě zda se uzel, ve kterém se na tuto otázku ptáme změní na uzlový list.

Gini impurity využíváme k výpočtu míry nečistoty v daném datasetu. Její hodnota se pohybuje mezi 0 a 1, kde 0 značí dokonalou čistotu, tedy že všechny vzorky v uzlu patří do jedné třídy.

Pokud máme C jakožto celkový počet tříd a p_i je pravděpodobnost vybrání datapointu s třídou, poté se gini impurity podle (Jairi, 2021), (James et al., 2021) vypočítá jako:

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

Pokud bychom tedy chtěli vypočítat giny impurity pro typy letadel, která jsou >= 5, která se oddělila v True větvi, tak bychom to vypočítali jako:

$$Gini = 1 - (Boeing^2 + Comac^2)$$

$$Gini = 1 - \left(\left(\frac{2}{3}\right)^2 + \left(\frac{1}{3}\right)^2\right)$$

$$Gini \approx 0{,}44$$

Dá se to tedy přirovnat k náhodnému losování. Kdyby nás někdo postavil před tři zavřené hangáry, a v každém by bylo letadlo společnosti Boeing, tak gini impurity nám říká, jak velká šance je, že ukážeme na jiný hangár, než na ten, ve kterém stojí letadlo společnosti Boeing, tedy gini impurity těchto tří hangárů je 0. Pokud by však v každém z těchto tří hangárů stálo jiné letadlo, tak šance, že ukážeme právě na ten, ve kterém stojí Boeing je 1/3, a tudíž gini impurity těchto tří hangárů je 1 – 1/3 = 0,66. (inspirováno: Gordon, 2017).

Entropie:

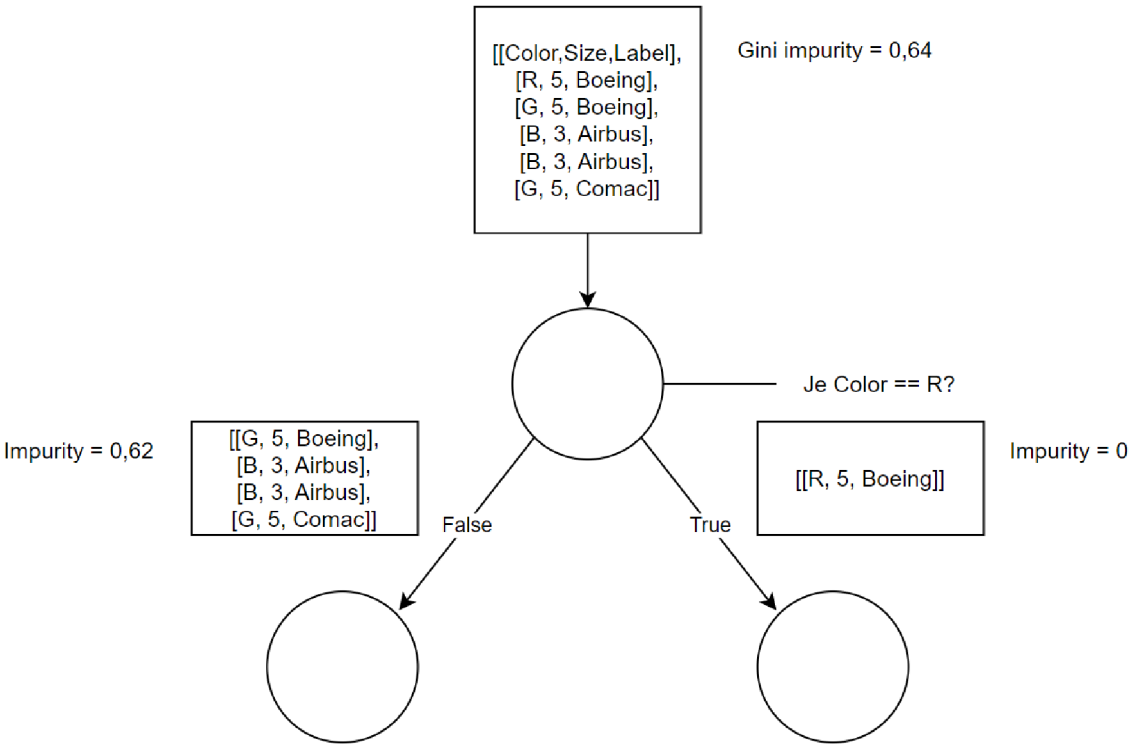
Entropie nám udává míru nejistoty v rozhodovacím stromu. Ačkoliv její využití není běžné v algoritmu CART, využívá se v dalších algoritmech rozhodovacích stromů, jako jsou ID3 nebo C4.5. (Dash, 2022)

$$E = -\sum_{i=1}^{n} p_i log_2(p_i)$$

Gini index:

Na obrázku 6 je jedno z vyhodnocení, které algoritmus musí provést, aby zjistil nejlepší rozdělení v rodičovském uzlu. Jelikož se jedná o takzvaný greedy algoritmus (Dash, 2022), tak tento algoritmus projde každý z parametrů datasetu, a vyhodnotí jaký z něj má giny index.

V tomto případě bude algoritmus vyhodnocovat giny index pro první parametr, kterým je červená (red – R) barva. Po rozdělení do dvou větví podle podmínky Color == R nám vzniknou dva datasety. Ve True větvi je už pouze jedno letadlo, tudíž impurity datasetu je 0. Ve větvi False nám zůstala čtyři letadla od třech různých společností, tudíž impurity je 0,62.



Obrázek 6: Ukázkový příklad pro výpočet information gain.
Inspirováno: (Gordon, 2017), vlastní zpracování.

Pokud bychom chtěli vypočítat gini index pro podmínku Color == R tak bychom museli vzít tyto dvě impurities z child uzlů (0,62 a 0) a udělat z nich průměrnou impurity (Gordon, 2017). Pro výpočet využijeme následující vzorec (Tao, 2020):

$$GINI(T) = \frac{\lfloor t_{left} \rfloor}{\lfloor T \rfloor} * Gini(t_{left}) + \frac{\lfloor t_{right} \rfloor}{\lfloor T \rfloor} * Gini(t_{right})$$

$$Avg.impurity = \frac{4}{5} * 0{,}62 + \frac{1}{5} * 0$$

$$Avg.impurity \approx 0{,}5$$

Nyní, když známe průměrnou impurity v child nodes, můžeme jí odečíst od rodičovské impurity, a získat tak gini index.

$$Gini\ index = 0{,}64 - 0{,}5$$

$$Gini\ index = 0{,}14$$

Na gini index bývá referováno jako „měřítko čistoty uzlu“. (James et al., 2021). Gini index může nabývat hodnot v rozmezí od 0 do 1. Gini index je založen na gini impurity a vyjadřuje rozdíl mezi gini impurity rodičovského uzlu a váženým průměrem gini impurity child uzlů. (Tao, 2020)

Pokud gini index nabývá hodnoty 0, pak to znamená, že nevede ke snížení nečistoty dat v porovnání s rodičovským uzlem. Čím nižší hodnota gini indexu je, tím menší pro nás takové rozdělení má přínos, respektive nám to říká, že obsahuje pozorování převážně z jedné třídy. (James et al., 2021). Čím více se však hodnota blíží 1, znamená to, že takové rozdělení vede k výraznému snížení nečistoty, a takové rozdělení je užitečné pro klasifikaci dat. (Tao, 2020)

Z obrázku 5 je zřejmé, že tato otázka znázorněná v obrázku 6 nám nepřinese nejlepší gini index. To v tento moment algoritmus ještě neví, a proto projde všechny zbylé otázky, a bude si držet informaci o té, která jeho informační zisk maximalizuje.

Listový uzel:

Listové uzly jsou uzly, které už jsou čisté, tudíž se na nich už dělá predikce. (Dash, 2022). Dá se říci, že to je typ uzlu, který už dál nejde rozdělit, protože všechny jeho parametry už nabývají stejných hodnot viz. obrázek 5, kdy máme v pravo dole v true větvi list, kde je sice 50% pravděpodobnost na předpověď Boeingu nebo Comacu, ale už jsme v našem datasetu vyčerpali všechny informace o tom, jak tyto dvě letadla od sebe odlišit. Listový uzel tedy reprezentuje výstup. (Sakkar, 2020)

Vytvoření stromu:

Rozhodovací stromy se staví rekurzivně, tedy při prvním volání dostane celý dataset. Po načtení datasetu ho celý projdeme, a najdeme nejlepší split question, tedy tu, která nám přinese nejvyšší gini index.

Na základě této otázky rozdělíme dataset na dvě podmnožiny tj. do dvou větví podle odpovědi na split otázku. True, False. Poté pro obě tyto větve rekurzivně zavoláme tu stejnou metodu, a proces se opakuje.

Pokud se dostaneme do stavu, kdy gini index uzlu je roven nule, poté tento uzel považujeme za list, a vracíme se do uzlu o krok výš. Pokud vyřešíme obě větve daného uzlu, dá se považovat za rozhodovací uzel, a opět se můžeme vrátit o krok výš. Tento proces se provádí tak dlouho, dokud se kořenový uzel nestane rozhodovacím uzlem. (Gordon, 2017)

2.3.3 Předzpracování dat

Data jsou kvalitní, pokud splňují požadavky zamýšleného využití. (Han et al., 2012). Podle (Han et al., 2012) existuje mnoho faktorů, které se vztahují ke kvalitě dat. Kvalita dat navíc může být subjektivní, jelikož různí uživatelé mohou kvalitu dat posuzovat různě.

Autoři ve výše zmíněné knize hovoří o tom, že v reálném světě se často setkáváme s neúplnými, nepřesnými a nekonzistentními daty, což ovlivňuje výslednou kvalitu datasetu. Kvalita datasetu se podle nich odvíjí od zamýšleného využití. Parametry, na základě kterých se dá taková kvalita datasetu posuzovat jsou: přesnost, úplnost, konzistence, aktuálnost, uvěřitelnost a lehká interpretovatelnost. Především první tři parametry jsou podle autorů problémem ve velkých databázích.

Předzpracování dat se podle autorů dělí na několik kroků, které jsou: čištění dat, integraci dat, redukci dat a transformaci dat. V rámci této práce postačí první zmíněná.

Čištění dat:

Čištění dat, neboli opravování datasetů zahrnuje vyplňování chybějících hodnot, jelikož datům chybí atributové hodnoty. Do čištění dat také patří vyhlazování šumových dat, což znamená opravování chyb nebo dat, která vybočují od očekávaných hodnot. Dále pak integrace dat, redukce dat a transformace dat.

Chybějící hodnoty:

Pokud některá hodnota chybí, je zde několik způsobů jakými se dá postupovat. Můžeme ignorovat n-tici. To se využívá převážně pokud chybí class label. Tato metoda je nejméně efektivní a není doporučována, protože můžeme odstranit další atributy, které mohou být užitečné.

Vyplnění chybějících atributů manuálně. Toto je nepraktické pro velké datasety s mnoha chybějícími hodnotami.

Využití globální konstanty. Tato metoda nahradí všechny chybějící hodnoty jako unknown, případně jinou proměnnou, která nemůže nastat. Tento přístup může vést k mylným závěrům při analýze dat.

Další metody pro práci s chybějícími hodnotami je nahrazují průměrem nebo mediánem nebo se snaží předpovědět chybějící hodnotu a doplnit jí. (Han et al., 2012)

Noisy data:

Šum je náhodná chyba nebo rozptyl měřené proměnné. (Han et al. 2012). Jedním ze způsobů jak šum odhalit je vizualizovat si data na grafech. Mezi techniky zpracování šumových dat patří „binning", který zkoumá okolí seřazených hodnot, které jsou rozdělené do jednotlivých „binů". Takto rozdělená data následně „uhlazuje". Na obrázku 7 je dataset cen, který je rozdělen do tří binů, a poté v rámci těch binů uhlazen. (Han et al., 2012)

Sorted data for *price* (in dollars): 4, 8, 15, 21, 21, 24, 25, 28, 34

Partition into (equal-frequency) bins:
Bin 1: 4, 8, 15
Bin 2: 21, 21, 24
Bin 3: 25, 28, 34

Smoothing by bin means:
Bin 1: 9, 9, 9
Bin 2: 22, 22, 22
Bin 3: 29, 29, 29

Smoothing by bin boundaries:
Bin 1: 4, 4, 15
Bin 2: 21, 21, 24
Bin 3: 25, 25, 34

Obrázek 7: Ukázka „binningu"
Zdroj: (Han et al., 2012)

Dalšími metodami pro zpracování šumu jsou lineární nebo multidimenzionální regrese a „outlier analysis", která dává podobné hodnoty do clusterů.

2.3.4 Chamtivost rozhodovacích stromů

Rozhodovací stromy jsou ze své přirozenosti tzv. chamtivé. (Maimon, 2006). Za chamtivost rozhodovacích stromů se označuje to, že si v uzlech vždy volí tu rozhodovací podmínku, která jim přinese největší informační zisk o datasetu, který do daného uzlu vstupuje.

S chamtivostí rozhodovacích stromů přichází problém, který se v angličtině nazývá „overfitting“.

2.3.5 Overfitting

(Smith a Koning, 2017) problém overfittingu ilustrují na příkladu, kdy ovoce s průměrem 2.87 inches klasifikujeme jako jablko, ale ovoce s průměry 2.86 a 2.88 inches klasifikujeme jako pomeranč.

Rozdíl mezi tím jak bychom my, jakožto lidé a počítač (v tomto případě rozhodovací stromy) klasifikovali data je úroveň detailu. My, jakožto lidé, bychom se eventuelně dostali do bodu, kde bychom s klasifikací přestali, protože bychom rozpoznali, že některé skupiny dat jsou moc náhodné na to, abychom je dále oddělovali, jelikož by to nebylo užitečné. Počítač se však nikdy nezastaví, a pokud nemáme nastavené žádné omezovací pravidlo tak nepřestane s klasifikací, dokud každý jeden kousek dat nebude rozdělen do kategorie, která je 100% čistá. (Smith a Koning, 2017)

Overfitting vzniká, pokud máme příliš rozsáhlý dataset z důvodu velkého počtu vstupních parametrů a celkového počtu trénovacích dat. To způsobuje, že rozhodovací strom ztrácí schopnost nová data generalizovat. V důsledku toho pak rozhodovací strom má velkou přesnost při predikci na trénovacích datech, ale už nižší přesnost na datech nových. (James et al., 2021)

2.3.6 Způsoby prořezání stromu

Pruning, neboli prořezání stromu, je jednou z možností jak řešit „overfitting“ strom. (James et al., 2021).

Nastavení minimálního počtu prvků pro každý list:

Prořezání může být řešeno už během růstu stromu, tomuto způsobu se říká „pre-pruning“ a jedním ze způsobů, jak ho dosáhnout, je, že nastavíme minimální počet prvků v každém listu. (Krueger et al., 2021)

Nalezení nejlepší hloubky stromu:

Pokud necháme růst strom dost dlouho, tak dojde k overfittingu. Dalším ze způsobů, jak tomuto problému předejít je, že nastavíme maximální hloubku stromu. Pokud však hloubka našeho stromu bude až moc nízká, může naopak dojít k underfittingu. Jedním z řešení je pro stejný dataset trénovat více stromů s různou hloubkou, a porovnávat jejich přesnost, a vybrat ten s nejvyšší mírou přesnosti. (Krueger et al., 2021)

Využití minimal cost complexity prořezání:

Pokud použijeme některou z předchozích možností a prořežeme strom před tím, než vůbec vznikne, tak sice můžeme získat menší strom, ale zároveň se můžeme připravit o možnosti, které by v této metodě mohly vytvořit dobré rozhodovací podmínky. Proto je lepší nejprve vytvořit velký strom T_0, a poté ho zpětně prořezat abychom získali podstrom. (James et al., 2021)

Scikit ve své knihovně sklearn.tree obsahuje knihovnu DecisionTreeClassifier, která má v sobě parametr ccp_alpha. Čím více se tento parametr zvyšuje, tím více je strom prořezaný, což zvyšuje nečistotu listů. (scikit/pruning)

Minimal cost complexity prořezání rekurzivně hledá nejslabší spojení. Toto nejslabší spojení je charakterizováno jako effective alpha, kde uzly s nejnižší effective alpha jsou prořezány první. (Krueger et al., 2021)

Než abychom zvažovali každý možný podstrom, zvážíme sekvenci stromů, které jsou indexovány jako nezáporný parametr ladění α. Pro každou hodnotu z α, která koresponduje s podstromem $T \subset T_0$.

Cost complexity measure pro strom T se dá vyjádřit jako:

$$R_\alpha(T) = R(T) + \alpha|T|$$

$R(T)$ - celková trénovací chyba na listových uzlech. V klasifikačních stromech se vypočítává z nečistoty.

$|T|$ – počet listových uzlů

α – complexity parametr (celé číslo)

Cost complexity prunning generuje sérii stromů, kde míra cost complexity pro podstrom T_t je:

$$R_\alpha(T_t) = R(T_t) + \alpha|T_t|$$

Parametr α redukuje složitost stromu tím, že kontroluje počet listových uzlů, což eventuelně snižuje míru overfittingu. (Krueger et al., 2021)

Obecný postup je tedy takový, že rekurzivně se pro každý uzel ve stromu vypočítá „cost complexity“, tedy hodnota, která nám říká, o kolik by celková chyba R(α) vzrostla, kdyby tento uzel byl odstraněn. Následně se identifikuje uzel s nejnižší „cost-complexity“ a je odstraněn ze stromu spolu se svými podstromy. Místo něj je vytvořen list.

Tento proces se opakuje pro různé hodnoty α a postupně odstraňuje uzly a zjednodušuje strom. Výsledkem je poté posloupnost stromů, z nichž každý odpovídá nějaké hodnotě α. V posledním kroku bude vybrán strom, jenž má nejlepší rovnováhu mezi chybami a složitostí za pomoci křížové validace. (James et al., 2021)

Dá se tedy říci, že se tato metoda snaží najít rovnováhu mezi složitostí stromu (jeho velikostí) a chybovostí stromu během testování.

Ačkoliv tato metoda není součástí knihovny CatBoost, které se tato práce primárně věnuje, je zmíněna, protože je součástí nejpopulárnější knihovny scikit.

2.3.7 Gradient boosting

Gradient Boosting Decision Tree (GBDT) je široce používaný algoritmus díky své efektivitě, přesnosti a interpretovatelnosti. (Ke et al., 2017) Podle Ke GBDT algoritmy dosahují „state-of-the-art“ výkonů v úlohách jako je klasifikace více tříd nebo předpověď kliknutí.

Gradient boosting je metoda, kterou lze přirovnat k učení se z chyb druhých. Na začátku se vytvoří jednoduchý rozhodovací strom, neboli „slabý žák“, který se naučí předpovědět cílové proměnné. Následně jsou do tohoto modelu přidáváni další slabí žáci, kteří se učí na reziduálních chybách, tedy rozdílech mezi skutečnými hodnotami a hodnotami předpovězenými předchozími slabými žáky. Cílem každé iterace je minimalizovat loss function, tedy míru toho, jak špatně model předpovídá data. Toho se dosáhne tím, že nové modely se učí předpovědět chyby předchozích modelů, čímž se postupně snižuje chyba celého souboru modelů. (Masui, 2022).

2.3.8 Další rozhodovací stromy

ID3:

ID3, celým názvem „Iterative Dichotomiser 3“ . Název vychází z toho, že tento algoritmus iterativně dichotomizuje (Sakkaf, 2020), tedy opakovaně rozděluje na dva. Algoritmus byl vytvořen Ross Quinlanem, a využívá „top-down“, „greedy“ přístup pro sestavení rozhodovacího stromu.

ID3 na rozdíl od algoritmu CART, který využívá Gini Index pro zvolení nejlepší rozhodovací otázky ID3 využívá Information gain, který se vypočítává z entropie, tedy ze snížení míry nejistoty v datech. (Sakkaf, 2020) ID3 je schopen pracovat pouze s kategorickými daty. (Scikit/tree)

C4.5:

Algoritmus C4.5 je nástupcem algoritmu ID3. Opět na něm pracoval Ross Quinlan, ve spolupráci s dalšími, a vytvořili algoritmus, který na rozdíl od svého předchůdce se nelimituje pouze na práci s kategorickými daty, ale je nyní schopen zpracovávat i data numerická. C4.5 rovněž vychází z informačního zisku a entropie. (Scikit/tree)

CHAID:

Algoritmus CHAID byl navržen v roce 1980 a je založen na testu chi-kvadrátu. Na rozdíl od dříve zmíněných algoritmů CHAID dokáže rozdělit uzel do více větví než pouze dvou jako to bylo u algoritmů ID3, C4.5 a CART. Dělení se

provede na tolik částí, kolik je kategorií ve vybrané proměnné. Díky těmto dělením je CHAID schopen zachytit více komplexní interakce. (Ramzain, 2020)

2.4 Python

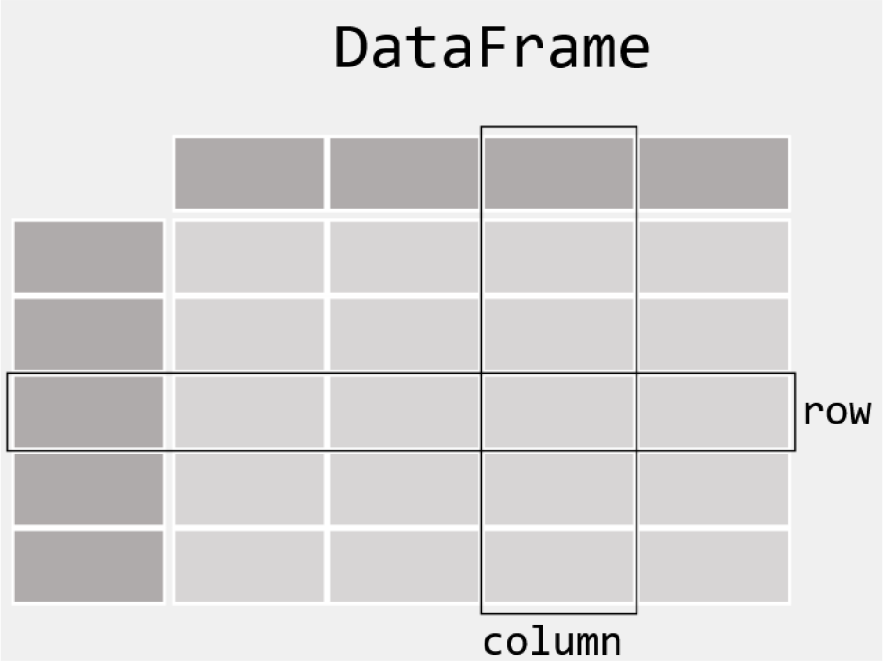
2.4.1 Pandas

Pandas je knihovna, která je open-source, a využívá se pro práci s datovými strukturami a často je nezbytnou součástí pro práci s dalšími knihovnami, které se využívají pro analýzu dat. (Pandas, 2023)

Níže je ukázka kódu pro import knihovny Pandas. Obvykle se pro tuto knihovnu používá v kódu zkratka pd.

```
import pandas as pd
```

Pandas slouží pro práci s tabulkovými daty, která běžně bývají uložena v Excelových tabulkách nebo v databázích. Pandas tyto tabulky načítá do objektu, který se nazývá DataFrame. DataFrame je dvoudimenzionální datová struktura se sloupci, které mohou obsahovat různé datové typy. Dá se to tedy představit jako například SQL tabulka. (Pandas)

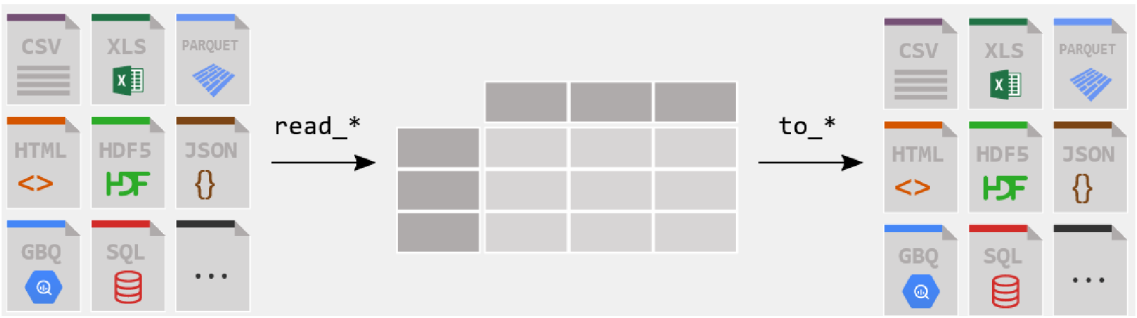


Obrázek 8: DataFrame
zdroj: (Pandas, n.d.).

Pandas podporuje práci s mnoha různými datovými formáty jako jsou například .CSV, .SQL a .JSON. Import těchto dat do DataFrame zajišťuje funkce Readeru s prefixem read_*, kde hvězdička značí formát importovaného souboru. Pro export slouží metoda Writeru to_* kde hvězdičkou označujeme formát, do kterého daný DataFrame chceme vyexportovat. (Pandas).

Níže je kód pro načtení CSV souboru se specifikací oddělovače.

```
flight_dataset = pd.read_csv("data/flightDataset.csv", sep=",")
```



Obrázek 9: Import a export datových souborů.
Zdroj: (Pandas, n.d.).

V Pandas můžeme z našeho celkového datasetu vybírat různé poddatasety na základě indexů nebo labelů. Prvek, který chceme vybrat se zapisuje do [] hranatých závorek. Funkce .loc nám umožňuje vybírat na základě labelů [‘a‘,‘b‘,‘c‘]. Funkce .iloc pak umožňuje vybírat na základě indexů v rozsahu od 0 do length-1. (Pandas).

Kód níže vybere všechna data kromě prvního sloupce.

```
data = data.iloc[:, 1:]
```

Pokud importujeme knihovnu matplotlib, můžeme jednoduše tato data vizualizovat za pomoci metody plot().

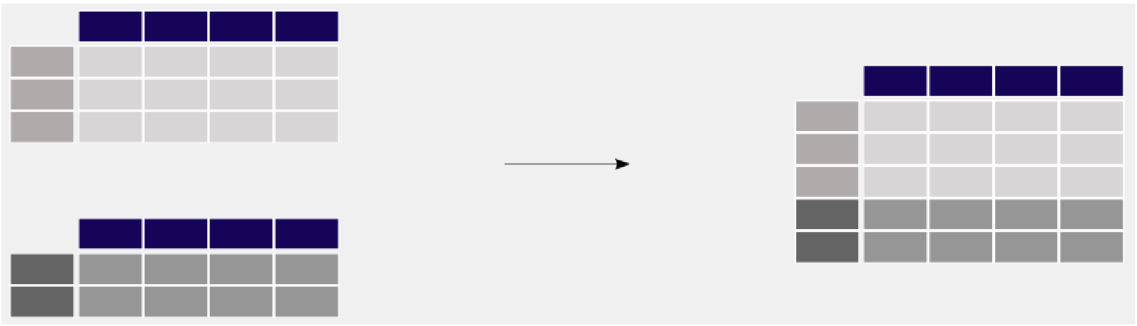
Pro přidání nového sloupce do stávajícího DataFrame napíšeme jméno daného DataFrame, a poté do [] hranatých závorek napíšeme název nového sloupce na levé straně přiřazení (Pandas). V kódu níže je uveden příklad, kdy přidáme nový sloupec hours do flight datasetu tím, že ze stávajícího sloupce seconds vypočítáme převod na hodiny.

```
flight_dataset["hours"] = flight_dataset["seconds"] / 3600
```

Knihovna Pandas má v sobě zároveň zabudované základní statistické funkce. Například pomocí funkce .mean() můžeme vypočítat průměr pro celý sloupec. Funkce .median() nám dá informaci o tom, jaký je medián zvoleného sloupce. Tyto a další operace se často provádějí v kombinaci s metodou groupby(), která se používá k seskupení dat v DataFrame na základě určitého kritéria. (Pandas).

Pandas také umožňuje spojování více tabulek za pomoci funkce concat(). Níže je kód pro spojení dvou DataFrames. (Pandas)

```
frames = [df1, df2]
result = pd.concat(frames)
```



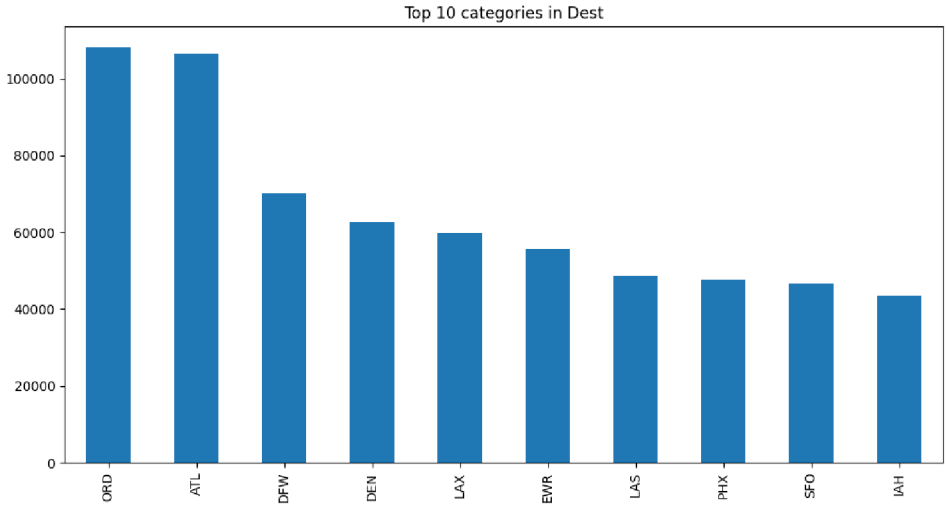
Obrázek 10: Sloučení dvou datasetů do jednoho pomocí funkce contac()
Zdroj: Pandas.

2.4.2 matplotlib

Vizualizace dat tvoří důležitou část podnikatelských aktivit, jelikož společnosti jich v dnešní době shromažďují velké množství a všechna tato data v sobě ukrývají pro podniky cenné informace, které mohu pomocí vizualizace dat lehce získat. (Shetty, 2018).

Matplotlib je 2-D vykreslovací knihovna, která pomáhá vizualizovat data. Matplotlib emuluje grafy a vizualizace podobné Matlabu, ale na rozdíl od Matlabu je zdarma a je jednoduchý na škálování a programování. (Shetty, 2018).

Na obrázku 11 je ukázka využití knihovny matplotlib, kdy okamžitě můžeme z datasetu o téměř dvou milionech řádcích identifikovat, jaká byla ve sledovaném období nejvíce navštěvovaná letiště ve Spojených státech.



Obrázek 11: Ukázka vizualizace pomocí Matplotlib
Zdroj: (vlastní zpracování).

2.4.3 Scikit-learn

Scikit-learn je open-source knihovna pro strojové učení v jazyce Python. Byla vytvořena v roce 2007 jako „Google Summer of Code project" Davidem Cournapeauem. V roce 2010 měl Scikit svůj první public release a od té doby k sobě přitahuje vývojáře z celého světa. (Scikit)

Scikit má přehlednou dokumentaci, tudíž je vhodný pro úplné začátečníky ve strojovém učení. Zároveň je kompatibilní s knihovnou CatBoost, tudíž zde budou zmíněny některé jeho užitečné funkcionality využité v této práci.

sklearn.model_selection:

train_test_split:

Funkce train_test_split slouží k rozdělení vstupního datasetu na trénovací a testovací množinu. Funkce na vstupu vezme dva DataFramy, případně jiný řetězcový datový typ, z nichž jeden obsahuje data bez popisků, a druhý jejich popisky. Dále jsou zde parametry train_size a test_size pro specifikaci toho, kolik procent celkového datasetu bude sloužit pro trénování a kolik pro testování dat. Pokud žádný z těchto dvou parametrů nebude zvolen, výchozí hodnota je 0,25 pro

test_size, a train_size se dopočítá, tedy bude mít hodnotu 0,75. Dalším parametrem je zde random_state, který slouží jako seed, a zajišťuje determiničnost rozdělení, tedy pokud zde vyplníme nějaké celé číslo, dostaneme na stejném datasetu se stejným poměrem trénovacích a testovacích dat vždy stejné rozdělení.

cross_val_score:

cross_val_score je další funkce z knihovny scikit, která hodnotí výkon algoritmu pomocí křížové validace. Tato funkce slouží ke křížové validaci, s jejíž pomocí lze lépe vyhodnotit účinnost modelu na neviděných datech. Pro nastavení slouží funkce cv, kdy hodnota v této funkci říká, na kolik foldů se dataset rozdělí. Následně vždy vybere jednu část, která bude sloužit jako testovací, a zbytek bude sloužit jako trénovací. Počet provedení je určen vstupním parametrem ve funkci cv. Každý z těchto foldů bude testovací právě jednou. (Scikit)

sklearn.metrics:

Funkce accuracy_score se využívá pro vyhodnocení výkonu klasifikačního modelu. Pomocí ní se vypočítává přesnost modelu, jakožto poměr správně předpovězených vzorků ku celkovému počtu. (Scikit)

2.4.4 CatBoost

CatBoost je poměrně nová knihovna (CatBoost releases) s verzí 1.0 zveřejněnou v roce 2021. Je to open-source knihovna vyvinutá firmou Yandex. Její název vychází ze spojení slov Category a Boosting. Knihovna CatBoost se zaměřuje na klasifikaci kategorických dat pomocí rozhodovacích stromů a využívá k tomu metody Gradient Boostingu. (CatBoost).

Podle autorů této knihovny (Prokhorenkova et al., 2019) dosahuje knihovna CatBoost lepších výsledků než další veřejně dostupné knihovny LightGBM a XGBoost. Toto tvrzení ověřovala studie (Hancock et al., 2020), která ve svém závěru říká, že CatBoost je dobrým kandidátem pro strojové učení zahrnujícím Big Data, a že výzkumníci by jeho využití měli zvážit pro heterogenní datasety, které obsahují kategorické proměnné.

Datové typy v CatBoost:

CatBoost podporuje numerické, kategorické, textové a embedding hodnoty. Pro kategorické hodnoty CatBoost provede před každým rozdělením převedení kategorických hodnot na numerické. Toto rozdělení se provádí pomocí statistických výpočtů na kombinacích kategorických vlastností a kombinacích kategorických a numerických vlastností.
Obecně tento převod kategorických hodnot na numerické zahrnuje:

1. Permutaci těchto objektů v náhodném pořadí.
2. Převod label hodnoty z floating pointu na integer na základě výběru loss funkce. V loss funkci určujeme, jaký problém řešíme. Určujeme, zda se jedná o regresi, klasifikaci nebo multiklasifikaci. Algoritmus na základě této specifikace bude data převádět.
3. Transformaci kategorických dat na numerická.

Trénování na GPU:

Jednou z předností knihovny CatBoost je, že umožňuje trénování na grafické kartě, což díky neustálému zvyšování výkonu grafických karet vede ke značně rychlejším výpočtům. Naproti tomu knihovna Scikit v současné době trénování na grafických kartách neumožňuje. (Scikit).

Podle webu weka (weka, 2021) je výhoda grafických karet ve strojovém učení ta, že grafické karty si mohou rozdělit operace na menší části, jelikož jsou složeny z tisíců menších jader, a poté tyto operace vykonávají současně bez nutnosti přepínání. To umožňuje provádět velké množství výpočtů najednou. V rámci algoritmu CART se takto může vypočítávat informační zisk pro všechny parametry v uzlu zároveň.

Kód pro nastavení tréninku na GPU:

```
from catboost import CatBoostClassifier
catboost_model = CatBoostClassifier(iterations=1000, task_type='GPU'
```

CatBoostClassifier:

CatBoostClassifier je třída, která slouží pro trénování a aplikování modelů na klasifikační problémy. (Catboost). Zároveň je kompatibilní s nástroji knihovny

scikit. V současnosti má tato třída 102 různých parametrů, tudíž přesnost a efektivita námi vytvořeného stromu záleží na tom, jaké parametry si zvolíme.

Boosting type:

Parametr boosting_type může nabývat dvě hodnoty: „Ordered“ a „Plain“. Ordered obvykle poskytuje lepší kvalitu v malých datasetech, jelikož podle (Prokhorenkova et al., 2020) zamezuje speciálnímu druhu „target leakage“, která je přítomna ve všech existujících implementacích gradient boosting algoritmů. Ve větších datasetech pak Ordered boosting způsobuje pomalejší učení a výrazně vyšší nároky na paměť. Defaultně je tento parametr nastaven na Ordered pouze pro učení na GPU pro datasety s méně než 50 000 objekty. (CatBoost)

3 Praktická část

3.1 Představení datasetu

Tento dataset byl získán z prostředí Kaggle (Gonzales, 2020) a obsahuje údaje o 1,93 milionu uskutečněných letů v období roku 2008 na území Spojených států. Každý řádek tohoto datasetu se skládá z 30 proměnných, z nichž první je samotné ID letu. Dataset je uložen ve formátu .CSV a jednotlivé hodnoty jsou oddělené čárkami. Jako zdroj dat pro tento dataset je uveden BTS.

Dataset není ideální, postrádá hromadu cenných informací, které byly zmíněny v příčinách zpoždění letů od (Tang, 2021) jako teplota, směr a rychlost větru, a další parametry, které by dále dokázaly zpřesnit předpověď, obzvláště po zmenšení datasetu v druhé části.

Proměnná ID je v rozmezí 0 – 7009727. Nebylo možné ověřit na základě čeho je tento rozdíl mezi počty letů, a přiřazováním ID způsoben, ale lze se domnívat, že původní dataset obsahoval i data o nevnitrostátních letech, která tuto nekonzistenci mezi počtem letů v datasetu a hodnotou v ID způsobila.

Proměnná Year je pro celý dataset stejná (2008). Na základě dalších sloupců Month (1-12) a DayOfMonth(1-31) lze říci, že tento dataset sleduje celý rok 2008.

Proměnná DepTime – skutečný čas odletu, CRSDepTime – plánovaný čas odletu, ArrTime – skutečný čas příletu a CRSArrTime – plánovaný čas příletu jsou ve formátu hhmm, a jejich hodnota vychází z lokálního času.

Proměnné UniqueCarrier – číslo dopravce, FlightNum – číslo letu, TailNum – označení letadla jsou další hodnoty, které poskytují údaje o konkrétním letu.

Proměnné ActualElapsedTime – skutečná délka letu, CRSElapsedTime – předpokládaná délka letu, AirTime – doba ve vzduchu, ArrDelay – o jak dlouhou dobu později letadlo přiletělo do cílové destinace a DepDelay – o jak dlouhou dobu později letadlo vzlétlo ze své startovní destinace, jsou proměnné, které jsou zaznamenány v minutách.

Proměnné Origin – letiště, ze kterého se vzlétá a Dest – cílové letiště jsou proměnné, které obsahují IATA kódové označení letiště.

Proměnná Distance obsahuje vzálenost letu v mílích.

Proměnná TaxiIn udává informace o tom, jak dlouho letadlo stráví na dráze po přistání, než se dostane na své parkoviště a proměnná TaxiOut informace o tom, kolik času letadlo stráví na zemi před vzhletem. Obě tyto hodnoty jsou uvedeny v minutách.

Proměnná Cancelled udává, zda byl let zrušen či nikoliv. Jeho hodnota je 0 nebo 1. CancellationCode obsahuje informace o tom, z jakého důvodu byl let zrušen. Může nabývat hodnot A = dopravce, B = počasí, C = NAS, D = security, N = let nebyl zrušen, tudíž neexistuje CancellationCode. Proměnná Diverted udává, zda byl let přesměrován na jiné letiště. Nabývá hodnot 0 nebo 1.

CarrierDelay – zpoždění leteckou společností, WeatherDelay – zpoždění z důvodu nepříznivého počasí, NASDelay – zpoždění způsobené národním leteckým systémem, SecurityDelay – zpoždění způsobené bezpečnostními důvody, LateAirCraftDelay – zpoždění způsobené zpožděním příchozího letadla, obsahují údaje o jednotlivých typech zpoždění v minutách a jsou více rozvedeny v teoretické části.

3.2 Příprava a čištění dat

Příprava a čištění dat probíhá v několika krocích. Nejprve je v samotném datasetu potřeba označit sloupec, který uživatel chce předpovědět jako „target“. V tomto případě je to proměnná ArrDelay.

Po označení této cílové proměnné můžeme načíst dataset.

```
def load_data(csv_file):
    data = pd.read_csv(csv_file, sep=",")
    return data
```

Po načtení datasetu dojde k odebrání prvního sloupce ID, jelikož nepřináší žádné informace o zpoždění letu, a na základě toho, jak rozhodovací stromy fungují, by tam mohly vzniknout nesmyslné rozhodovací podmínky. Například by zde mohly být zakódovány jiné, na první pohled neviditelné informace, jako například údaj o čase. Tedy kdyby se zpoždění tvořily například o dni Díkuvzdání,

protože více lidí cestuje, mohl by určitý rozsah ID sloužit jako informace o zpoždění, ačkoliv by to nesly jiné sloupce, jako je měsíc a den.

```
def remove_id_column(data):
    data = data.iloc[:, 1:]
    return data
```

Jelikož je cílem předpovědět to, zda let bude zpožděný, tak dojde k převedení target sloupce z časových hodnot na bool hodnoty, aby mohla proběhnout klasifikace. To se dělá na základě podmínky, zda je let zpožděn o více než patnáct minut. Pokud ano, jedná se o zpožděný let.

```
def binarizer(y, minutes):
    y_binarized = (y > minutes).astype(int)
    return y_binarized

```

Po převodu se v datasetu vyskytuje 8397 řádků, které neobsahují hodnotu pro ArrDelay, tedy zvolený target. Jelikož nemáme informace o tom z jakého důvodu nenabývají hodnoty a zabraňují klasifikaci, tak v počtu téměř dvou milionů uskutečněných letů nepředstavují významnou část datasetu a tyto řádky jsou odstraněny.

```
def remove_rows_with_missing_targets(data, target_col):
    data = data.dropna(subset=[target_col])
    return data
```

Dalším důležitým krokem, spíše z hlediska algoritmu než samotného datasetu, je detekce cat features, tedy kategorických vlastností. Z toho důvodu je z datasetu nutné vybrat jeden řádek, který obsahuje všechny proměnné tak, aby detekce proběhla správně. Alternativně by se tam všechny sloupce mohly zadat manuálně, ale tím by se snížila znovupoužitelnost. Jelikož hledání takového řádku je běžný krok při analýze datasetu, jeví se jako lepší řešení.

```
def find_cat_features(data, row_index=0):
    selected_row = data.iloc[row_index]

    cat_features = [col for col in data.columns if isinstance(
    selected_row[col], str)]

    return cat_features
```

Po předchozích úpravách zde pořád zůstaly tři řádky, kterým chybí kategorická proměnná, jelikož však měly hodnotu pro ArrTime, byla jim chybějící hodnota změněna na „unknown“.

```
def fill_missing_categorical_values(data, cat_features):
    for cat_feature in cat_features:
        data[cat_feature].fillna('unknown', inplace=True)
        return data
```

3.3 Rozdělení dat na tréninkovou a testovací množinu

Nejprve se datasetu oddělí cílová proměnná, tedy dataset se rozdělí na dvě části. Část x obsahuje všechna data kromě sloupce target a část y pak pouze sloupec target.

Pro další rozdělení je použita funkce knihovny scikit train_test_split, která rozdělí dataset na trénovací a testovací v poměru, který je dán parametrem test_size. Parametr random_state pak zajišťuje to, že výběr dat bude konzistentní.

```
def split_data(data, test_size, random_state):
    x = data.drop('target', axis=1)
    y = data['target']
    x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=test_size, random_state=random_state)
    print("splitting data")
    return x_train, x_test, y_train, y_test
```

3.4 Implementace algoritmu CART pomocí CatBoost

Pro vytvoření modelu byla použita knihovna CatBoost a její třída CatBoostClassifier.

```
def train_model(x_train, y_train, cat_features, model_path=None):
    catboost_model = CatBoostClassifier(
    learning_rate=0.03,boosting_type="Plain", iterations=1000,
    loss_function='Logloss', depth=16, task_type='GPU', verbose=50)

    catboost_model.fit(x_train, y_train, cat_features=cat_features)
```

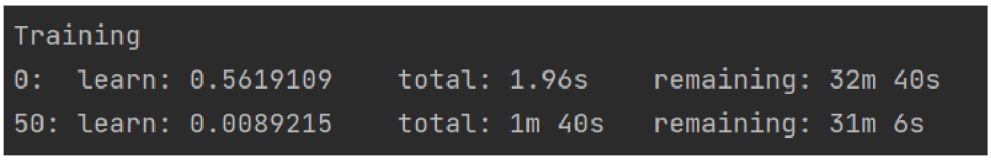
Výše je uvedeno vytvoření instance CatBoostClassifier, kde learning_rate je ponechána defaultně na 0,03 jelikož nevykazovala nějaký větší vliv na přesnost

modelu. boosting_type je nastaven na „Plain“, zde nebylo možné porovnat výkonnost modelu s modelem, který využívá boosting_type „Ordered“, protože pro takto velký dataset nebylo možné zajistit dostatek paměti. V dokumentaci je však uváděno, že boosting_type „Ordered“ je vhodný pro menší datasety do 50 tisíc záznamů, což tento dataset s téměř dvěma miliony údajů o uskutečněných letech není.

Počet iterací je zde nastaven na 1000, jelikož se jedná o defaultní hodnotu, a také z důvodu maximalizace přesnosti. Nevýhodou tohoto nastavení je fakt, že výsledný model s počtem iterací 1000 má přes 1GB velikosti. Model, který má pouze 100 iterací má pouze 100MB, takže by se zde dalo mluvit o nějaké přímé úměře. loss_function je nastavena jako „Logloss“, jelikož je vhodná pro binární klasifikaci.

Hloubka je nastavena na 16, jelikož je to maximální hloubka, kterou CatBoost umožňuje a pro takto velký dataset je adekvátní. Jako task_type byl zvolen GPU, jelikož grafické karty jsou pro tento typ úloh vhodnější než CPU díky svému velkému množství jader, která mohu provádět mnoho výpočtů najednou.

Zde je ukázka porovnání čtyřjádrové i7 6700k a RTX 3060 s 12GB RAM. Na prvním řádku obou obrázků 12 a 13 je počáteční odhadovaný čas ve sloupci remaining. Na tom druhém pak odhadovaný zbývající čas po padesáti iteracích.
Na obrázku 12 je zobrazen zbývající čas pro trénink na GPU. Na obrázku 13 je zobrazen trénink na CPU.



Obrázek 12: Ukázka délky trénování na GPU
Zdroj: (vlastní zpracování).

Za zmínku u obrázku 13 stojí to, že po prvních 50 iteracích se odhadovaná délka ještě o více než 50% zvýšila.

```
Training
0:  learn: 0.5726987    total: 3.2s remaining: 53m 18s
50: learn: 0.0096894    total: 4m 42s   remaining: 1h 27m 42s
```

Obrázek 13: Ukázka délky trénování na CPU
Zdroj: (vlastní zpracování).

Z obrázků výše je jasně vidět, že GPU si vede přibližně čtyřikrát lépe než CPU.

3.5 Modelování a výsledky

Pro vyhodnocení modelu je vytvořena funkce evaluate_model, která nejprve pomocí metody predict udělá předpovědi na natrénovaném modelu za pomoci testovacích dat. Poté tyto predikce pomocí accuracy_score z knihovny scikit porovná s reálnými hodnotami a vypočte přesnost.

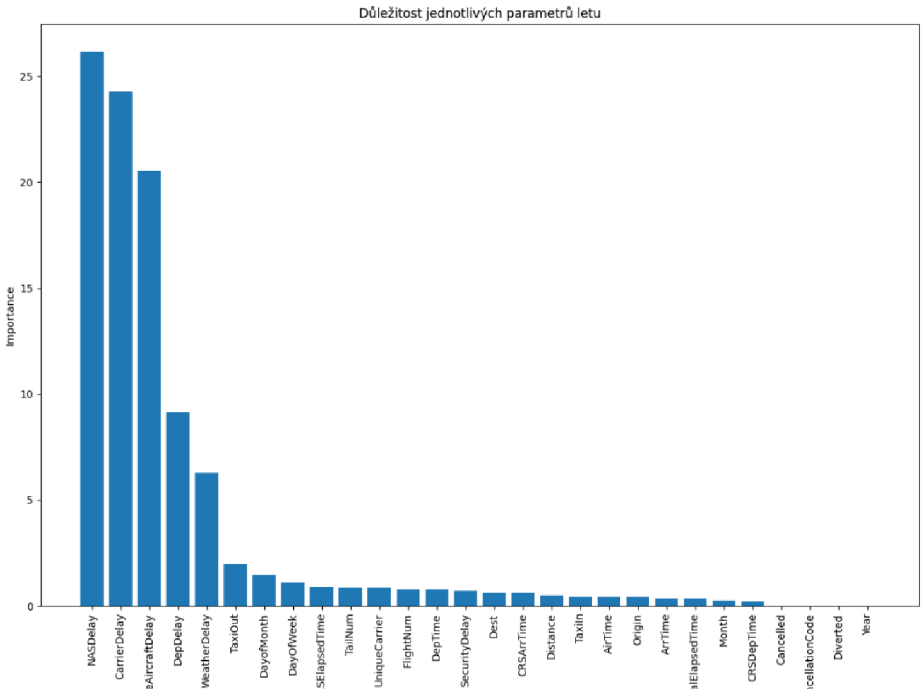
```python
def evaluate_model(catboost_model, x_test, y_test):
    y_pred = catboost_model.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy
```

3.6 Diskuze a interpretace výsledků

Po testování na zbylých 20% vstupního datasetu model na obrázku 14 vykazuje 99.988% přesnost predikce při 1000 iteracích. Ze znalosti datasetu a ze znalosti grafu důležitosti jednotlivých parametrů na obrázku 15 by se dalo říci, že natrénovaný model předpovídá zpoždění letu na základě dílčích zpoždění.

```
Evaluating
CatBoost model accuracy: 99.9878136%
```

Obrázek 14: Přesnost pro hloubku 16 při 1000 iteracích
Zdroj: (vlastní zpracování).



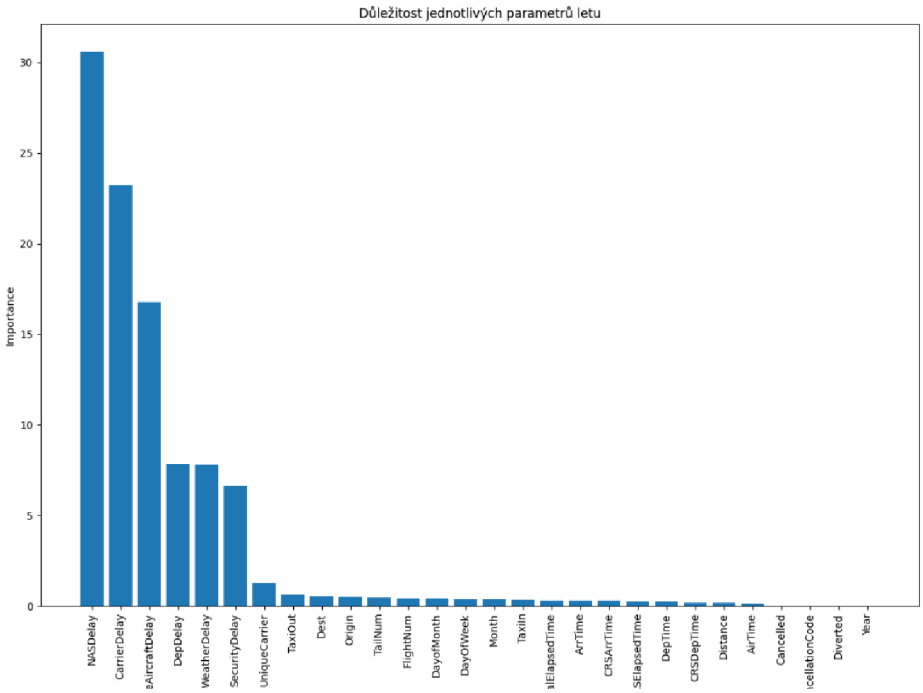
Obrázek 15: Důležitost jednotlivých parametrů při 1000 iteracích
Zdroj: (vlastní zpracování).

Ve srovnání s modelem s 1000 iteracemi má model se 100 iteracemi na obrázku 16 jen nepatrně horší přesnost, nicméně velikost uloženého modelu je desetkrát nižší.

```
Evaluating
CatBoost model accuracy: 99.9821093%
```

Obrázek 16: Přesnost pro hloubku 16 při 100 iteracích
Zdroj: (vlastní zpracování).

Zároveň má na obrázku 17 jiné rozložení důležitosti parametrů kdy dává větší důraz na zpoždění způsobené NAS.



Obrázek 17: Důležitost při 100 iteracích hloubky 16
Zdroj: (vlastní zpracování).

V tabulce 1 můžeme lépe porovnat hlavní parametry obou modelů, které s měnícím se množstvím iterací mění důležitost jednotlivých parametrů, a zároveň složitosti modelů, které je reprezentovány velikostí modelu.

Počet iterací	Přesnost	Velikost modelu	Klíčové parametry		
			NAS delay	Carrier delay	Aircraft delay
1000	99.9878%	1077 MB	26%	24%	20%
100	99.9821%	102 MB	30%	23%	16%

Tabulka 1: Srovnání stejného datasetu při 100 a 1000 iteracích.
Zdroj: (vlastní zpracování).

3.7 Vizualizace dat

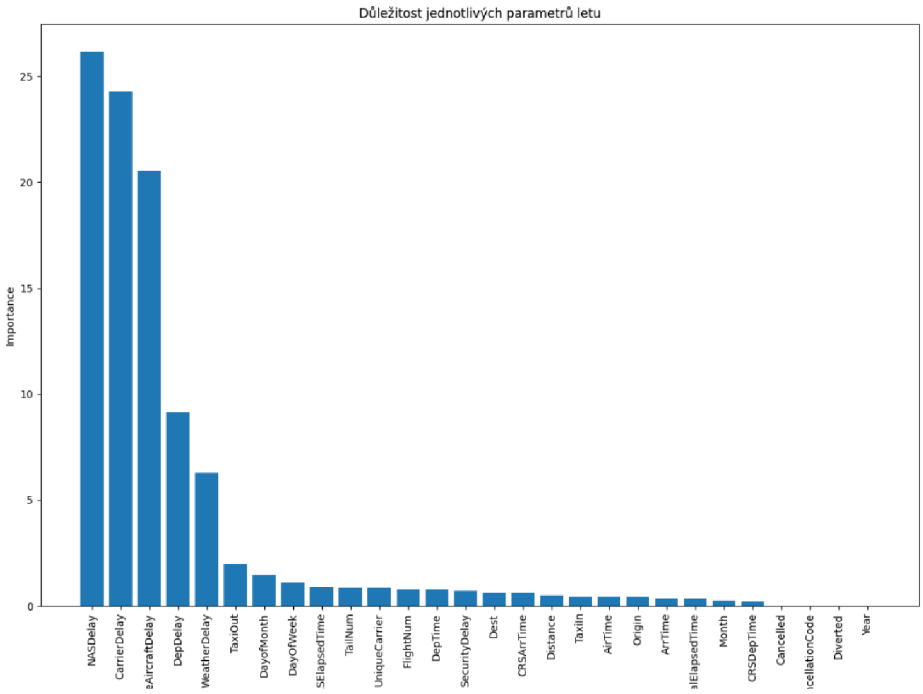
Pro získání hodnot v grafech důležitosti byla využita funkce get_feature_importances knihovny CatBoost, z které po vizualizaci v obrázku 18 vyplývá, že nejdůležitějším parametrem pro predikci zpoždění letů v tomto modelu je zpoždění způsobené národním leteckým systémem (NAS), které přesahuje 25%. To zároveň také znamená, že změny v tomto parametru budou mít největší vliv na výslednou předpověď.

```
def plot_feature_importances(catboost_model, x_train):
    importances = catboost_model.get_feature_importance()
    sorted_indices = importances.argsort()[::-1]
    feature_names = x_train.columns[sorted_indices]
    plt.figure(figsize=(15, 10))
    plt.bar(range(x_train.shape[1]), importances[sorted_indices],
    align='center')
    plt.xticks(range(x_train.shape[1]), feature_names,
    rotation='vertical')
    plt.title("Důležitost jednotlivých parametrů letu")
    plt.xlabel("Feature")
    plt.ylabel("Importance")
    plt.show()
```

Dalším významným parametrem je zpoždění způsobené leteckými společnostmi, které se pohybuje lehce pod 25%. Třetím významným parametrem pro predikci zpoždění letů je zpoždění příchozího letu, které se pohybuje okolo 20%.

Tyto tři zpoždění jsou tedy nejdůležitějšími pro předpověď zpoždění letů, protože společně tvoří přibližně 70%. Následovány jsou dalšími proměnnými, které také poskytují informace o dílčích zpožděních - zpožděném odletu a zpoždění způsobeném nepříznivým počasím.

Dále lze z grafu v obrázku 18 vyčíst, že hodnoty jako rok, odklonění letu nebo zrušení letu na předpověď zpoždění letu mají žádný nebo minimální vliv.

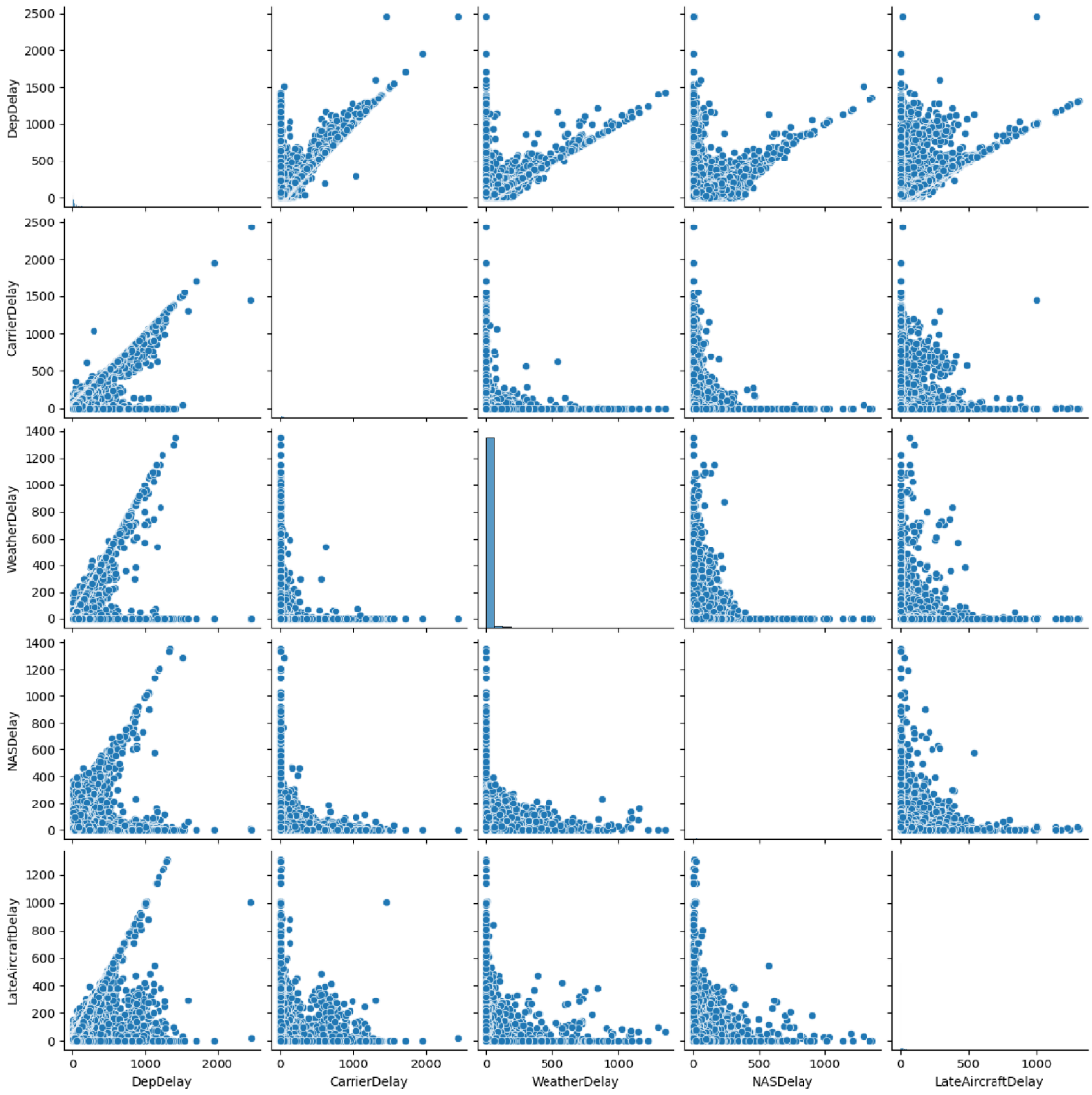


Obrázek 18: Důležitost jednotlivých parametrů při 1000 iteracích
Zdroj: (vlastní zpracování).

Nyní se na základě pohledu na graf důležitosti jednotlivých parametrů v obrázku 18 nabízí otázka, proč je důležitost zpoždění odletu letu tak nízká v porovnání s většinou ostatních zpoždění, když zpožděný odlet by měl mít přímý vliv na přílet.

```
def plot_scatter_matrix(data, columns):
    selected_data = data[columns]
    sns.pairplot(selected_data)
    plt.show()
```

Výše uvedený kód slouží pro vytvoření matice korelace na základě vybraných proměnných. V tomto případě DepDelay, CarrierDelay, WeatherDelay, NASDelay, LateAircraftDelay.



Obrázek 19: Matice proměnných
Zdroj: (vlastní zpracování).

Z grafu na obrázku 19 vyplývá, že mezi zpožděním odletu (DepDelay) a ostatními typy zpoždění existuje korelace. Na základě znalosti algoritmu CART, kdy víme, že dělá rozhodnutí, která mu poskytnou nejvyšší informační zisk by dávalo smysl, že mu zpoždění odletu, které je součtem dílčích zpoždění přinese nejlepší informace. Jelikož však máme nastavenou hloubku 16, což je dost na to aby se projevily i dílčí zpoždění a CatBoost je schopen zachytit komplexní interakce a učit se z chyb tak jsou upřednostněny dílčí proměnné, které se s množstvím iterací upřesňují, což je vidět v obrázku 15, kdy se důležitost NAS zpoždění při 1000

iteracích snížila o 4% oproti obrázku 17 při 100 iteracích a více se dal důraz na ostatní proměnné.

3.8 Modifikace datasetu

Výsledky výše mohou na první pohled vypadat skvěle. Dokážeme říci, že let bude zpožděn s přesností hraničící se sto procenty. To ale není úplně pravda. Spíše to znamená to, že algoritmus byl schopen pochopit vzorce jako „Pokud je NASDelay vyšší, než 15 minut, bude let pravděpodobně zpožděn o více než 15 minut, tudíž nedorazí do své destinace včas.“ Toto je však uplatnitelné pouze tehdy, pokud například letadlo už stojí na ranveji, a nemůže kvůli NAS vzlétnout.

Práce s předchozím datasetem nepůsobí zajímavě, protože se nedá moc o předpovědi hovořit, spíše ověřuje to, že pokud má model všechna data, dokáže je správně interpretovat a použít. Proto nyní budou odebrány ty sloupce, které poskytují nějakou informaci o časovém zpoždění, která nemohla být dopředu známa. Aby si například osoba, která má v šest večer letět na dovolenou do Turecka už v 10 ráno mohla říci, že její let bude pravděpodobně zpožděn nebo aby se letiště už dopředu na takovéto situace mohlo připravit.
Pro tyto účely byly odebrány všechny tyto sloupce:

```
['Diverted', 'AirTime', 'ArrTime', 'DepTime','TaxiIn', 'TaxiOut',
'ActualElapsedTime', 'CarrierDelay','DepDelay',
'WeatherDelay','NASDelay','SecurityDelay','LateAircraftDelay']
```

V datasetu zůstaly pouze časy s odhadovaným odletem a příletem, které se běžně objevují na letenkách.

Model byl opět trénován pro 1000 a 100 iterací, aby bylo možné pozorovat rozdíly ve zvolených parametrech. Dle očekávání se snížila přesnost, jelikož na vstupu nyní chybělo mnoho časových údajů, podle kterých se lehce dalo zpoždění identifikovat.

Rozdíl v přesnosti modelů byl nyní 3%, tedy vyšší než u modelů, které byly trénovány na celém datasetu, a přesnost byla oproti modelům trénovaným na celém datasetu nižší o více než 30%. V případě 1000 iterací to bylo o 31,51% a v případě 100 iterací o 34,52%. Tabulka 2 obsahuje data o obou nově natrénovaných modelech.

Počet iterací	Přesnost	Velikost souboru
1000	0,68476	3473 MB
100	0,65464	102 MB

Tabulka 2: Nově natrénované modely pro 1000 a 100 iterací.
Zdroj: (vlastní zpracování).

Co však bylo neočekávané je to, že nový model se po odebrání 13 sloupců z datasetu při 100 iteracích nezmenšil, ale zůstal velikostně prakticky stejný, a u 1000 iterací se dokonce více než třikrát zvětšil.

Počet iterací	Velikost modelu	
	Celý dataset	Zmenšený dataset
1000	1077 MB	3473 MB
100	102 MB	102 MB

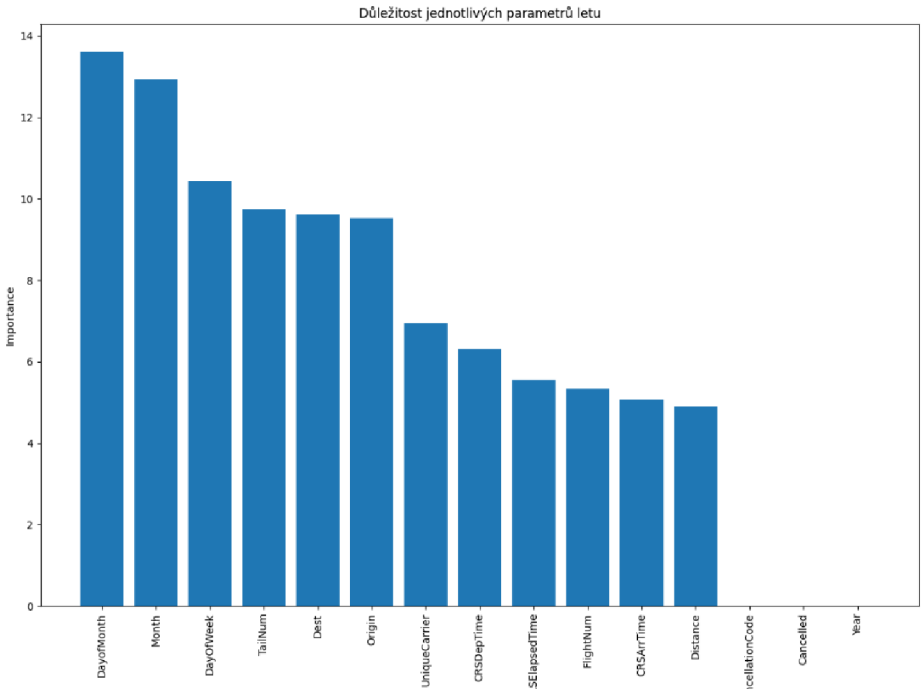
Tabulka 3: Porovnání velikostí natrénovaných modelů
Zdroj: (vlastní zpracování).

V tabulce 4 jsou zleva seřazeny klíčové parametry dle jejich důležitosti. Přesnější přehled je poté zachycen na grafech. U modelu při 100 iteracích převládají měsíc, dopravce a den v měsíci následovány počátečním a cílovým letištěm a dnem v týdnu.

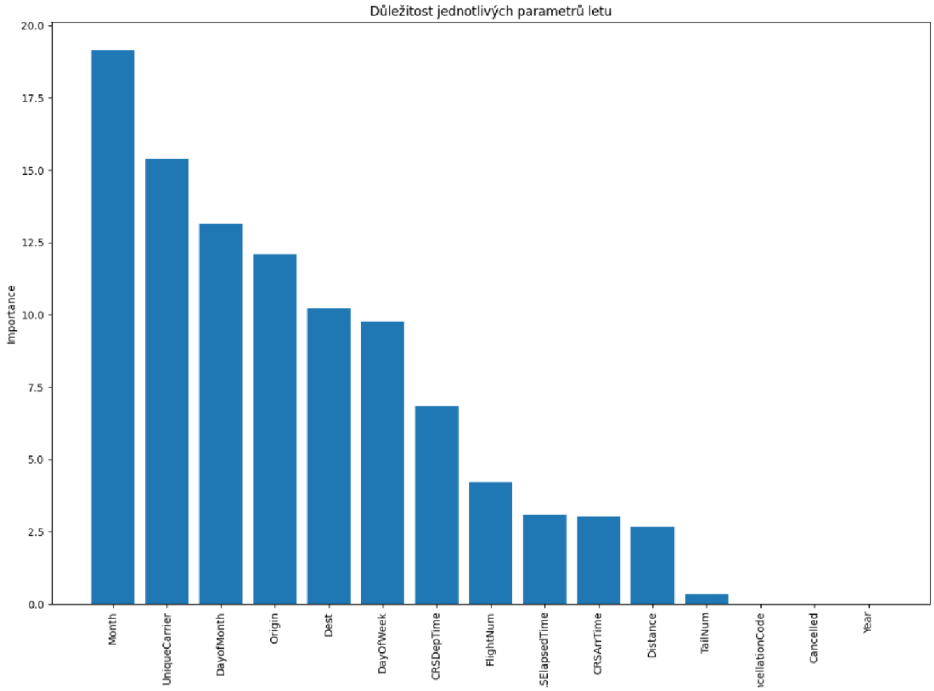
U 1000 iterací to jsou den v měsíci, měsíc a den v týdnu, tedy při takovém množství iterací si už model zvládl udělat rozdělení na konkrétní dny. Následuje pak přesnější označení konkrétního letadla, cílová destinace a místo vzletu. Poslední tři hodnoty jsou zajímavé, protože mají téměř stejnou hodnotu.

Počet iterací	Klíčové parametry dle důležitosti					
1000	DayofMonth	Month	DayOfWeek	TailNum	Dest	Origin
100	Month	UniqueCarrier	DayOfMonth	Origin	Dest	DayOfWeek

Tabulka 4: seřazení parametrů dle důležitosti pro zmenšený dataset
Zdroj: (vlastní zpracování).



Obrázek 20: Graf důležitosti při 1000 iteracích.
Zdroj: (vlastní zpracování).

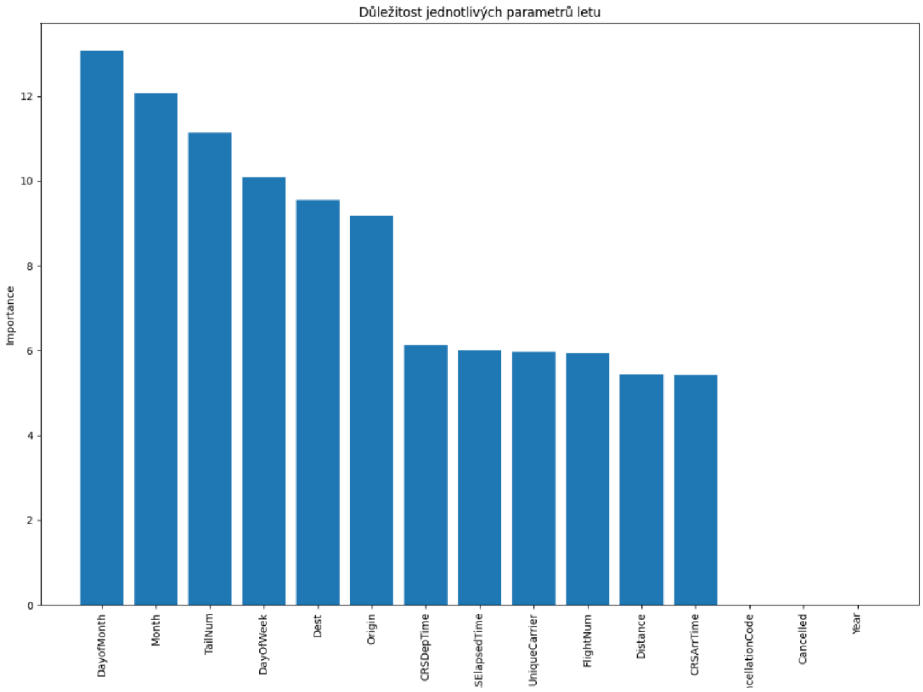


Obrázek 21: Graf důležitosti při 100 iteracích
Zdroj: (vlastní zpracování).

Pro ověření zvýšení velikosti modelu byl pro stejné parametry a zmenšený dataset vytvořen ještě jeden model s 2000 iteracemi.

Počet iterací	Přesnost	Velikost modelu
2000	68,78978	7391 MB

Tabulka 5: Ověření velikosti pro 2000 iterací
Zdroj: (vlastní zpracování).



Obrázek 22: Graf důležitosti při 2000 iteracích.
Zdroj: (vlastní zpracování).

Oproti obrázku 20 můžeme v obrázku 22 pozorovat větší rozložení důležitosti z předních hodnot na ty další, prohození pořadí důležitostí, malé zvýšení přesnosti předpovědi o 0,3% a ustálování nižších hodnot důležitosti na podobné úrovni.

3.9 Ověření s datasetem z výchozí studie

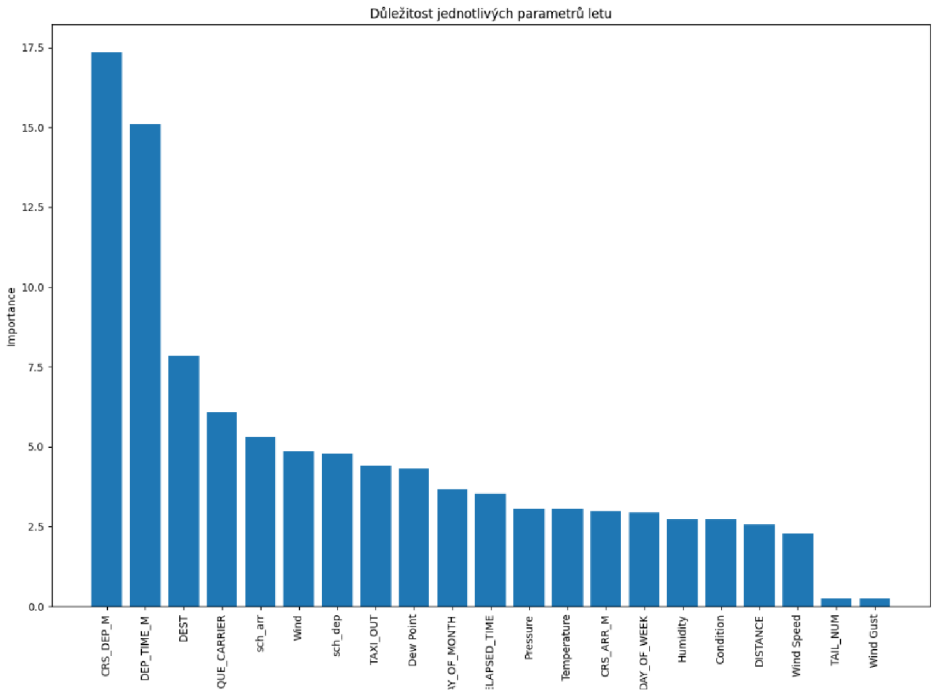
V článku (Tang, 2021), ze kterého bylo vycházeno při výběru algoritmu strojového učení je poskytnut dataset, na kterém byly algoritmy trénovány pro předpověď zpoždění odletu. Tento dataset na rozdíl od předchozího datasetu

obsahuje informace o různých přírodních vlivech jako je směr a rychlost větru, které dále upřesňují předpověď zpoždění letu.

Algoritmus	Přesnost
Random Forest	92,40%
Gradient Boosted Tree	93,34%
CatBoost GBDT	94,86%
Decision Tree	97,78%

Tabulka 6: Porovnání přesnosti CatBoost s dalšími algoritmy využívajících stromy
Zdroj: (Tang, 2021, vlastní zpracování).

V tabulce 6 je porovnání tří algoritmů založených na rozhodovacích stromech z výchozího článku s algoritmem CatBoost využitým v této práci. Implementace CatBoost v této práci dosáhla o 1,52% vyšší přesnosti než alternativní algoritmus gradient boostingu. Algoritmus rozhodovacího stromu byl však pořád přesnější.



Obrázek 23: Důležitost při 1000 iteracích na datasetu z článku
Zdroj: (vlastní zpracování).

V obrázku 23 je vidět, že dataset, který byl využit v článku (Tang, 2021) obsahuje sloupce CRS_DEP_M – plánovaný čas odletu a DEP_TIME_M – skutečný čas odletu, které mají v modelu nejvyšší rozhodovací váhu. Stejně jako v první části, kdy

dataset ještě nebyl zmenšen, máme v datasetu nějaké číselně vyjádřené zpoždění, což pak vede v případě předpovědi zpoždění letů k vysoké přesnosti předpovědi. Je zde opět nějaký jednoduchý vztah, v tomto případě rozdíl dvou hodnot, které nám dávají informaci o zpoždění odletu.

4 Shrnutí výsledků

V první části, kdy byl model trénován na kompletním datasetu s téměř dvěma miliony řádků o třiceti sloupcích tento algoritmus dokázal najít v datech vzory, které vedly i při pouhých sto iteracích, což je desetkrát méně, než je výchozí používaná hodnota, k téměř 100% přesnosti předpovědi.

Ve druhé části, kdy už tato předpověď byla „čtením z letenkových lístků“ se nám při 2000 iteracích podařilo dostat až na 68,79%, což už je téměř 7 z 10 správně předpovězených zpoždění jen na základě údajů vyčtených z letenky.

Ve třetí části byl algoritmus porovnán s algoritmy v (Tang, 2021) článku. Zde si vedl trochu lépe, než náhodný les i gradient boosted tree. Obyčejný rozhodovací strom byl však na tomto konkrétním datasetu o zhruba 3% přesnější. Oproti algoritmům, které nebyly založeny na rozhodovacích stromech si vedl v této klasifikační úloze ještě lépe.

5 Závěry a doporučení

Bakalářská práce se zabývá problematikou zpoždění letů z hlediska jejich předpovědí. Zpoždění letů způsobuje každý rok leteckému odvětví jen ve Spojených státech náklady v desítkách miliard dolarů, a proto jakýkoliv způsob jejich včasné identifikace je důležitou součástí jejich eliminace.

Teoretická část je zaměřena na seznámení se s problematikou zpoždění letů a jejich příčinami. Je zde detailně popsán algoritmus CART (Classification and Regression Tree), který je v této práci využíván. Dále zde jsou popsány Python knihovny a jejich třídy, které jsou součástí této práce.

Praktická část se věnuje předpovědím na velkém datasetu, který čítá dva miliony záznamů o vnitrostátních letech na území Spojených států. První dataset obsahuje všechny údaje a předpověď na něm dosahuje téměř 100% přesnosti. Jeho modifikovaná verze neobsahující údaje o dílčích zpožděních, která má svými dostupnými informacemi blíže k něčemu prakticky využitelnému má přesnost předpovědi okolo 68%, tedy téměř 7 z 10 správně identifikovaných zpoždění.

V závěru praktické části byl zvolen testovací dataset, který porovnával výsledky s implementací různých druhů rozhodovacích stromů, a zde si vedl trochu lépe, než alternativní implementace Gradient boosted algoritmu, čímž se ověřila jeho správná implementace.

Práci by bylo možné rozšířit například vytvořením mobilní aplikace pro cestující nebo webové aplikace pro letecké společnosti, která by přistupovala k údajům o letech, které by se na serveru v nějakých intervalech, například jeden den, zapracovávaly do modelu a cestující nebo letecké společnosti by se mohly dotazovat na svůj let.

6 Seznam použité literatury

[1] Alpaydin, E. Introduction to machine learning. 3rd ed. Cambridge, Massachusetts: MIT Press, 2014. ISBN 978-0-262-02818-9

[2] SMITH, Chris a KONING, Mark. *Decision Trees and Random Forests: A Visual Introduction For Beginners*. California, (California): CreateSpace Independent Publishing Platform, 2017. ISBN 978-1549893759.

[3] TANG, Yuemin. *Airline Flight Delay Prediction Using Machine Learning Models* [online]. Singapore: ACM, October 15-17, 2021 [cit. 2023-04-25]. Dostupné z: doi:https://dl.acm.org/doi/fullHtml/10.1145/3497701.3497725

[4] KHAKSAR, H. a A. SHEIKHOLESLAMI. Airline delay prediction by machine learning algorithms. *Scientia Iranica* [online]. 2017, 0-0 [cit. 2023-03-20]. ISSN 2345-3605. Dostupné z: doi:10.24200/sci.2017.20020

[5] BALL, Michael, Cynthia BARNHART, Martin DRESNER, et al. *Total delay impact study: a comprehensive assessment of the costs and impacts of flight delay in the United States* [online]. United States Department of Transportation, 2010 [cit. 2023-03-21]. Dostupné z: https://rosap.ntl.bts.gov/view/dot/6234

[6] *Airline On-Time Statistics and Delay Causes* [online]. [cit. 2023-03-21]. Dostupné z: https://www.transtats.bts.gov/ot_delay/OT_DelayCause1.asp?20=

[7] *Types of Delay* [online]. Federal Aviation Administration [cit. 2023-03-13]. Dostupné z: https://aspm.faa.gov/aspmhelp/index/Types_of_Delay.html

[8] RASCHKA, Sebastian a Vahid MIRJALILI. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Third Edition. Packt Publishing, 2019. ISBN 978-1-78995-575-0.

[9] JAMES, Gareth, Daniela WITTEN, Trevor HASTIE a Robert TIBSHIRANI. *An introduction to statistical learning: with applications in R*. Second edition. New York: Springer, [2021]. Springer texts in statistics. ISBN 978-1071614174.WOLFF, Rachel. *5 Types of Classification Algorithms in Machine Learning* [online]. August 26th, 2020 [cit. 2023-03-26]. Dostupné z: https://monkeylearn.com/blog/classification-algorithms/

[10] JAIRI, Idriss. *Gini Gain vs Gini Impurity | Decision Tree — A Simple Explanation* [online]. Dec 19, 2021 [cit. 2023-03-27]. Dostupné z: https://medium.com/@jairiidriss/a24ebfeebee9

[11] GORDON, Josh. *Machine Learning Recipes #8: Let's Write a Decision Tree Classifier from Scratch* [online]. Google Developers, 2017 [cit. 2023-03-27]. Dostupné z: https://youtu.be/LDRbO9a6XPU

[12] DASH, Shailey. Decision Trees Explained: Entropy, Information Gain, Gini Index, CCP Pruning. *Towards Data Science* [online]. Nov 2, 2022 [cit. 2023-03-27]. Dostupné z: https://towardsdatascience.com/4d78070db36c

[13] TAO, Christopher. Get Your Decision Tree Model Moving by CART. *Towards Data Science* [online]. Sep 13, 2020 [cit. 2023-03-27]. Dostupné z: https://towardsdatascience.com/82765d59ae09

[14] SAKAF, Yaser. Decision Trees: ID3 Algorithm Explained. *Towards Data Science* [online]. Mar 21, 2020 [cit. 2023-03-29]. Dostupné z: https://towardsdatascience.com/89df76e72df1

[15] scikit. *Decision Trees* [online]. [cit. 2023-03-29]. Dostupné z: https://scikit-learn.org/stable/modules/tree.html

[16] MAIMON, Oded a Lior ROKACH. *Data Mining and Knowledge Discovery Handbook*. New York, NY: Springer, 2006. ISBN 978-0-387-25465-4.

[17] KRUEGER, Edward, Sheetal BONGALE a Douglas. *Build Better Decision Trees with Pruning: Reducing Overfitting and Complexity of Decision Trees by Limiting Max-Depth and Pruning* [online]. Jun 14, 2021 [cit. 2023-03-30]. Dostupné z: https://towardsdatascience.com/8f467e73b107

[18] scikit. *Post pruning decision trees with cost complexity pruning* [online]. Cambridge, MA: Scikit-learn [cit. 2023-03-30]. Dostupné z: https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html

[19] Pandas. *Pandas documentation* [online]. b. r. PyData [cit. 2023-04-01]. Dostupné z: https://pandas.pydata.org/docs/index.html

[20] CatBoost. [online]. b. r. [cit. 2023-04-01]. Dostupné z: https://catboost.ai/en/docs/

[21] PROKHORENKOVA, Liudmila, Gleb GUSEV, Aleksandr VOROBEV, Anna Veronika DOROGUSH a Andrey GULIN. *CatBoost: unbiased boosting with categorical features* [online]. Moskva, 20 Jan. 2019, 23 [cit. 2023-04-03]. Dostupné z: https://arxiv.org/abs/1706.09516

[22] HANCOCK, John T. a Taghi M. KHOSHGOFTAAR. CatBoost for big data: an interdisciplinary review. *Journal of Big Data* [online]. 2020, **7**(1) [cit. 2023-04-03]. ISSN 2196-1115. Dostupné z: doi:10.1186/s40537-020-00369-8

[23] KE, Guolin, Qi MENG, Thomas FINLEY, Taifen WANG, Wei CHEN, Weidong MA, Qiwei YE a Tie-Yan LIU. *LightGBM: A Highly Efficient Gradient Boosting Decision Tree* [online]. [cit. 2023-04-03]. Dostupné z: https://www.semanticscholar.org/paper/497e4b08279d69513e4d2313a7fd9a55dfb73273

[24] HAN, Jiawei, Micheline KAMBER a Jian PEI. *Data mining: concepts and techniques*. 3rd ed. Waltham: Morgan Kaufmann, c2012. Morgan Kaufmann series in data management systems. ISBN 978-0-12-381479-1.

[25] *Weka: Why GPUs for Machine Learning? A Complete Explanation* [online]. September 10, 2021 [cit. 2023-04-09]. Dostupné z: https://www.weka.io/learn/hpc/gpus-for-machine-learning/

[26] Matplotlib. *Matplotlib 3.7.1 documentation* [online]. 2023 [cit. 2023-04-09]. Dostupné z: https://matplotlib.org/stable/plot_types/index.html

[27] SHETTY, Badreesh. Data Visualization using Matplotlib. *Towards Data Science* [online]. Nov 12, 2018 [cit. 2023-04-09]. Dostupné z: https://towardsdatascience.com/16f1aae5ce70

[28] RAMZAI, Juhi. *Simple guide for Top 2 types of Decision Trees: CHAID & CART* [online]. Jun 19, 2020 [cit. 2023-04-09]. Dostupné z: https://towardsdatascience.com/8695e441e73e

[29] VERA, Adrián. *Flight Delay EDA (Exploratory Data Analysis)* [online]. Kaggle, Nov 28, 2017 [cit. 2023-04-11]. Dostupné z: https://www.kaggle.com/code/adveros/flight-delay-eda-exploratory-data-analysis

[30] GONZALES, Giovanni. *Airlines delay: Airline on-time statistics and delay causes* [online]. 2020 [cit. 2023-04-23]. Dostupné z: https://www.kaggle.com/datasets/giovamata/airlinedelaycauses

[31] MASUI, Tomonori. *All You Need to Know about Gradient Boosting Algorithm: Algorithm explained with an example, math, and code* [online]. Jan. 20, 2022 [cit. 2023-06-13]. Dostupné z: https://towardsdatascience.com/2520a34a502

[32] Catboost releases. *Catboost releases* [online]. [cit. 2023-06-15]. Dostupné z: https://github.com/catboost/catboost/releases

7 Přílohy

1) Elektronická příloha - .zip archiv s Jupyter notebookem obsahujícím zdrojový kód.

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Zdeněk Archleb**
Studium:	I2000327
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Predikce zpoždění letů metodou strojového učení**
Název bakalářské práce AJ:	Flight Delay Prediction Using Machine Learning

Cíl, metody, literatura, předpoklady:

Cílem práce je popsat vybranou klasickou metodu strojového učení (rozhodovací stromy) a realizovat vlastní aplikaci (předpověď zpoždění letu).

1.Úvod

2. Teoretická část

2.1. Strojové učení

2.2. Rozhodovocí stromy

2.3. Python

3. Praktická část

3.1. Předpovídání zpoždění letů

3.2. Návrh modelu

3.3. Implementace

3.4. Testování

4. Výsledky

5. Závěr

- https://www.kaggle.com/datasets/jimschacko/airlines-dataset-to-predict-a-delay
- Mokhtarimousavi,S., Mehrabi,A. (2022) Flight delay causality: Machine learning technique in conjunction with random parameter statistical analysis. Int. J. of Transportation Science and Technology, https://doi.org/10.1016/j.ijtst.2022.01.007.

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	doc. RNDr. Kamila Štekerová, Ph.D., MSc.
Oponent:	Ing. Karel Mls, Ph.D.
Datum zadání závěrečné práce:	15.10.2021