FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Interpretabilita modelů strojového učení
Student:	Bc. Jakub Štercl
Vedoucí:	Ing. Mgr. Jan Romportl, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2020/21

Pokyny pro vypracování

1. Seznamte se s problematikou interpretovatelnosti netransparentních modelů strojového učení.
2. Proveďte rešerši a vzájemné porovnání aktuálně dostupných metod (např. LIME, SHAP, atd.) a nástrojů či knihoven (např. IML, DALEX, atd.) pro interpretaci modelů strojového učení.
3. Zvolte si vhodný experimentální dataset, natrénujte na něm dostatečně netransparentní model (např. CNN) a proveďte analýzu interpretovatelnosti jednou zvolenou metodou.
4. Navrhněte a implementujte softwarový nástroj (např. sada dokumentovaných skriptů v Pythonu), který v prostředí Data Science Pipeline větší společnosti usnadní rutinní validaci produkčně nasazovaných modelů z hlediska jejich interpretability vůči businessové logice a vůči požadavkům férového strojového učení.
5. Použití tohoto nástroje ilustrujte na konkrétním případu vhodně zvoleného reálně fungujícího modelu v komerčním prostředí.

Seznam odborné literatury

Lundberg S. M., Lee S.: A Unified Approach to Interpreting Model Predictions. In Advances in Neural Information Processing Systems 30 (NIPS 2017). http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions

Ribeiro M. T., Singh S., Guestrin C.: "Why Should I Trust You?": Explaining the Predictions of Any Classifier. https://arxiv.org/abs/1602.04938

Molnar C.: Interpretable Machine Learning. Online book https://christophm.github.io/interpretable-ml-book/

Barocas S., Hardt M., Narayanan A.: Fairness and Machine Learning. Incomplete work-in-progress book https://fairmlbook.org

Informal compendium of online resources for machine learning interpretability https://github.com/jphall663/awesome-machine-learning-interpretability

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 8. května 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Interpretabilita modelů strojového učení

Bc. Jakub Štercl

Katedra Softwarového Inženýrství
Vedoucí práce: Ing. Mgr. Jan Romportl, Ph.D

8. ledna 2020

Poděkování

Chtěl bych poděkovat nejen svému vedoucímu Ing. Mgr. Janu Romportlovi, Ph.D, za jeho cenné rady a připomínky při vedení práce, ale také celému Big Data týmu společnosti O2 Czech Republic a.s, zejména pak Tomášovi Kalendovi, Petru Stanislavovi a Kataríně Vlčkové, za jejich podporu a pomoc s prací.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. ledna 2020 .....

Odkaz na tuto práci

Štercl, Jakub. *Interpretabilita modelů strojového učení*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Interpretabilita modelů strojového učení slouží k vysvětlení chování, jinak mnohdy zcela neprůhledných, modelů. Tato práce představuje nejpoužívanější metody a nástroje interpretability. Vlastnosti jednotlivých metod pak ilustruje na několika příkladech, jak jednoho modelu z praxe, tak ukázkových. Praktická část práce se pak zabývá návrhem a implementací softwarové knihovny ExpyBox, která slouží pro usnadnění práce a experimentování s vybranými metodami interpretability v prostředí interaktivních Jupyter notebooků.

Klíčová slova interpretabilita, strojové učení, Jupyter, Python

Abstract

Interpretability of machine learning models tries to explain behavior of unknown models. This thesis shows the most frequently used methods and tools for interpretability. It ilustrates the behavior of selected methods on a few examples of artificial and actual productionalized models. Second part of the work is focused on design and implementation of ExpyBox package, which aims to provide easy interface for working and experimenting with chosen methods of interpretability in interactive Jupyter notebooks.

Keywords interpretability, explainability, machine learning, Jupyter, Python

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Oblast umělé inteligence a strojového učení je v dnešní době jednou z nejrychleji se rozvíjejících oblastí informatiky. Nové modely strojového učení stále více zasahují do života běžných lidí, průmyslové výroby, ale třeba i rozhodování a strategického plánování podniků. A právě s rostoucími nároky roste i komplexita těchto modelů, kvůli které se mnohdy stávají neprůhlednými a jejich rozhodovací procesy pro člověka téměř nepochopitelnými. Proto je nezbytné mít nástroje, jejichž pomocí můžeme pochopit, proč a na základě čeho se určitý model rozhoduje tak, jak se rozhoduje.

Vzhledem k aktuálnosti tématu existuje, či dokonce vzniká, řada metod jak interpretabilitu modelů zajistit. Nicméně pro efektivní využití těchto metod v praxi je nutné se v této rychle se měnící oblasti zorientovat. Proto si úvodní část této práce klade za cíl čtenáři přiblížit nejpoužívanější metody interpretability modelů strojového učení, jejichž použití a vlastnosti následně ilustruje na několika příkladech.

Práce se také věnuje návrhu a implementaci knihovny ExpyBox – knihovny značně usnadňující tvorbu interaktivních Jupyter notebooků pro vybrané metody interpretability jako např. LIME nebo SHAP. Interaktivní notebook totiž pomáhá urychlit experimentování s různými parametry metod, pro dosažení optimálních výsledků.

Kapitola 1

Cíl práce

Cílem práce je seznámit se s používanými metodami pro interpretabilitu modelů strojového učení a tyto metody porovnat. Následně na základě tohoto srovnání navrhnout a implementovat softwarový nástroj, který pomůže zefektivnit validaci produkčně nasazovaných modelů vůči businessové logice a také požadavkům na férové strojové učení.

V teoretické části práce se tedy nejprve zaměřím na samotnou interpretabilitu modelů strojového učení, její vlastnosti, přínosy apod. Poté provedu představení a srovnání dostupných metod, knihoven, či nástrojů interpretability. Použití vybraných metod zároveň ilustruji na ukázkových, uměle vytvořených, příkladech.

Pomocí získaných znalostí a zkušeností pak navrhnu a implementuji softwarový nástroj pro metody interpretability, jehož použití již ilustruji na konkrétním, prakticky používaném, modelu společnosti O2 Czech Republic a.s.

Kapitola 2

Interpretabilita modelů strojového učení

2.1 Strojové učení

Pro pochopení interpretability modelů strojového učení je samozřejmě nutné mít alespoň základní představu o tom, co takové strojové učení je. V této práci nebudu zacházet do detailu, pouze rámcově představím, případně připomenu, co se pod tímto pojmem skrývá.

2.1.1 Definice

Podobně jako u jiných konceptů lze nalézt mnoho různých, ale přesto podobných definic, přičemž žádnou nelze označit za tu jedinou správnou. Dalo by se říci, že strojové učení je snaha přimět počítače, aby se učily a chovaly jako lidé, aby se samy v průběhu času učily a zlepšovaly pomocí získaných, případně dodaných, dat a zkušeností[1].

Typickým příkladem může být program, který odhaduje cenu automobilu na základě jeho vlastností. Pokud použijeme některý z algoritmů strojového učení, pak se program například na základě předchozích prodejů naučí určité vzorce chování ceny v závislosti na parametrech vozu a podle nich bude u dalších automobilů odhadovat cenu.

2.1.2 Kategorizace algoritmů strojového učení

Nejčastěji bývají algoritmy strojového učení rozdělovány do čtyř hlavních kategorií – *supervised learning* (učení s učitelem), *unsupervised learning* (učení bez učitele), *semi-supervised learning* (kombinace učení s učitelem a bez učitele) a *reinforcement learning* (zpětnovazebné učení)[2].

2.1.2.1 Supervised learning

Už český překlad *učení s učitelem* naznačuje, že se jedná o algoritmy, které se učí na základě vstupních dat, u kterých je určen očekávaný výsledek. Cílem je vytvořit nějakou funkci, která namapuje vstupní proměnné (*features*) na výstupní proměnnou. Supervised learning algoritmy se tedy snaží *modelovat vztahy a závislosti mezi vstupy a výstupní proměnnou*[2].

Pokud je výstupní proměnná diskrétní, hovoříme o *klasifikaci*. Jedná se vlastně o rozřazení do tříd, např. „spam“ a „not spam“. Pokud je výstupem spojitá veličina, jedná se o *regresi*[3]. Jako příklad regrese může opět posloužit výše uvedené odhadování ceny automobilu. Pokud se algoritmus bude učit na základě uskutečněných prodejů z historie, pak se jedná o klasický příklad supervised learning regrese.

2.1.2.2 Unsupervised learning

Pokud, na rozdíl od supervised learning, nemáme v trénovacích datech nějakou zadanou hodnotu výstupu a model se tedy snaží nalézt vztahy a vzorce v datech bez znalosti správné odpovědi, řadíme algoritmus do skupiny *unsupervised learning* algoritmů. Tyto algoritmy nemohou být přímo použity pro regresi nebo klasifikaci, jelikož předem nevíme, jak bude vypadat výstup a nelze proto model naučit standardním způsobem[4]. Místo toho se používají například pro *clustering* – podobně jako u klasifikace se jedná o rozdělení vstupního datasetu do tříd, nicméně toto rozdělení probíhá čistě na základě podobnosti instancí. Metody pro clustering mají také tendenci přeceňovat podobnost mezi jednotlivými záznamy. Dále se unsupervised learning algoritmy používají například pro detekci anomálií nebo autoencoding.

2.1.2.3 Semi-supervised learning

Semi-supervised learning je jakýsi hybrid mezi supervised a unsupervised learning. Část trénovacích dat má známou hodnotu výsledku, nicméně využívají se i záznamy, které ji nemají. Taková situace může nastat například pokud je zjištění správné hodnoty náročné (vyžaduje zásah lidského experta). V takovém případě se ukazuje, že i neohodnocené záznamy zvyšují, při použití správných metod, přesnost natrénovaných modelů.

2.1.2.4 Reinforcement learning

Poslední kategorií algoritmů strojového učení je tzv. reinforcement learning. Tento pojem označuje algoritmy, nebo modely, které se učí na základě interakce s okolím. Od začátku mají daný pouze cíl, např. výsledný stav prostředí nebo maximalizaci počtu získaných bodů, a na základě provedených akcí dostávají odměny nebo naopak tresty. Podobně jako když učíme domácí

mazlíčky reagovat na povely pomocí dobrot, se tedy agent naučí vykonávat správné akce vedoucí k cíli a vyhýbat se těm nesprávným.

Reinforcement learning algoritmy dosahují velmi dobrých výsledků a dokáží, s dostatkem učení, překonávat člověka v oblastech, které byly donedávna považovány za počítačem nezvládnutelné. Například *AlphaGo*[5], které poráží nejlepší lidské hráče hry Go nebo *OpenAI Five*[6], která 13.4.2019 porazila mistry světa v počítačové hře DOTA 2.

2.2 Interpretabilita modelů strojového učení

2.2.1 Definice interpretability

Interpretabilita nemá, a nejspíše ani nemůže mít, přesnou matematickou definici. V přirozeném jazyce znamená slovo interpretovat *srozumitelně vysvětlit* nebo *vyložit*. Interpretabilitu obecně tedy můžeme definovat jako schopnost vysvětlit nebo prezentovat data ve formě, která je pochopitelná pro člověka[7].

Interpretabilitu modelů strojového učení pak můžeme definovat jako *míru, se kterou dokáže pozorovatel pochopit důvod rozhodnutí*[8] nebo jako míru, *se kterou dokáže pozorovatel správně a efektivně předvídat výsledek metody*[9].

Osobně se přikláním spíše k první zmíněné definici, jelikož pokud člověk dokáže předvídat výsledek, pak automaticky chápe i důvod, proč k němu model dospěl, ale pro interpretabilitu, podle mého názoru, není nutné umět výsledek *předvídat*.

2.2.2 Důležitost interpretability

K čemu ale taková interpretabilita je a proč bychom se jí měli zabývat? Proč modelům jednoduše nevěříme a neptáme se *proč* učinily právě takové rozhodnutí?

Pravdou je, že ne pro všechny modely a jejich aplikace je interpretabilita bezpodmínečně nutná. Někdy nám stačí vědomí, že model má dostatečnou accuracy (přesnost). To platí zejména v případech, kdy chyba modelu nemá žádné závažné důsledky (např. doporučovací systémy v e-shopech) nebo je problém natolik dobře prostudovaný, zanalyzovaný a řešení je ověřené, ideálně v reálných aplikacích, že rozhodnutí můžeme beze strachu věřit. Tak tomu je například u problému optického rozpoznávání znaků (OCR)[7].

Existují ale i modely, u kterých je potřeba, nebo minimálně velmi vhodné, ptát se i *proč* zrovna takové rozhodnutí. Kim a Doshi-Velez[7] tvrdí, že tato potřeba vyplývá z neúplnosti ve formalizaci problému, tedy v otázce, kterou modelu pokládáme. Vlastně se jedná o problémy, u kterých nám úplně nestačí jenom jednoduchá odpověď, ale hodnotu přináší i vysvětlení. To koresponduje se způsobem, jakým člověk přistupuje k učení a k chápání světa. Pokud výstupem modelu bude predikce, že pacient do půl roku zemře, bude pro nás mít velkou hodnotu i *proč*, abychom tomu mohli u dalších lidí předcházet.

Podobně pro mě nebude mít velkou hodnotu předpověď, že „mi zítra nebude dobře“ ve srovnání s prognózou, že „mi zítra nebude dobře, protože jsem vypil šest panáků rumu a osm piv“ – vysvětlení mi dává možnost naučit se nechtěnému stavu předcházet.

Dalším opodstatněním interpretability jsou modely, jejichž rozhodnutí mohou mít velký dopad, ať už na život jednotlivce, či na firmu. Takových modelů se v praxi nasazuje čím dál více, ať už se jedná o hodnocení úvěruschopnosti (credit scoring) v bankovnictví, či modely doporučující výstavbu nových vysílacích věží telekomunikační společnosti apod. U takových modelů je interpretabilita důležitým faktorem kontroly a je správné se ptát i *proč*. Co se aplikací zásadně ovlivňující jedince týče, nárok na takové vysvětlení (tzv. „right to explanation“ – právo na vysvětlení) je dokonce, alespoň částečně[10], zakotveno ve směrnici Evropské Unie označované „General Data Protection Regulation“ (GDPR)[11].

Kromě kontroly ale zvyšuje interpretabilita také důvěru v samotný model. Pro člověka je mnohem snazší důvěřovat modelu, který dokáže svoje výstupy odůvodnit. Podobně mu snadněji přiřadí „lidské“ vlastnosti, díky čemuž mu dokonce odpustí i případnou chybu.

2.2.3 Kategorizace metod interpretability modelů

Jednou z vlastností, podle které můžeme metody interpretability dělit je forma jejich výstupu[12].

- Souhrnné statistiky proměnných – mnohé metody poskytují statistiky pro každou proměnnou. To může být jedno číslo určující její důležitost, nebo komplexnější výsledek, popisující například i vztahy mezi jednotlivými dvojicemi proměnných apod. Výstup těchto metod může většinou být vizualizován. Pro některé metody, např. partial dependence (částečná závislost) proměnné, dokonce dává smysl pouze vizualizace samotná.

- Vnitřní hodnoty modelu – takovou interpretaci bychom mohli označit za *vlastní* (Christoph Molnar[12] ji označuje jako *intrinsic*). Tyto metody jsou z principu specifické pro konkrétní model. Příkladem mohou být váhy v lineárních modelech nebo samotná struktura rozhodovacího stromu. Za zmínku stojí i to, že někdy se stírá hranice mezi vnitřními hodnotami modelu a statistikami proměnných, jelikož např. u lineárních modelů jsou váhy jak statistiky, tak vnitřní hodnoty.

- Datový bod – do této kategorie spadají metody, které pro to, aby model učinily interpretovatelným, využívají nový, nebo i stávající, datový bod (instanci). Jednou z těchto metod je tzv. counterfactual explanations (vysvětlení protipříkladem). Tato metoda pro vysvětlení predikce pro danou instanci nalezne instance podobné, pozměněné tak, aby výstup modelu byl zásadně jiný (např. jiná predikovaná třída).

- Interpretovatelný model – jednoduchým řešením, jak interpretovat vlastně jakýkoli black-box model (model, jehož vnitřní strukturu neznáme), je aproximovat jej pomocí jiného, snadno interpretovatelného modelu. Ten je pak interpretován buď pomocí vnitřních hodnot nebo statistikami proměnných.

Dále je možné rozdělit metody na modelu specifické a agnostické. Jak napovídá název, jedná se o rozlišení, zda je metoda použitelná pouze pro konkrétní model, příp. skupinu modelů, a nebo ji lze použít kdykoli. Mezi specifické metody patří všechny metody, jejichž výstupem jsou vnitřní hodnoty modelu, např. již zmíněné váhy v lineárních modelech.

Agnostické metody lze využít k interpretaci jakéhokoli modelu a používají se tedy na již hotový a natrénovaný model. Z principu musí tedy k modelu přistupovat jako k černé skříncе (black box), přičemž jej většinou analyzují čistě na základě analýzy dvojic vstup–výstup.

Určitě existuje ještě mnoho možností, jak metody pro interpretabilitu modelů dělit, už proto, že se jedná o rychle se rozvíjející odvětví, nicméně pro základní dělení se mi zmíněné zdá dostatečné.

2.2.4 Rozsah interpretability

2.2.4.1 Globální interpretabilita

Globální interpretabilita znamená, že jsme schopni vysvětlit celkové chování modelu. Můžeme ji rozdělit na holistickou a modulární. Holistická spočívá v porozumění, jak se model rozhoduje, založeném na celkovém pohledu na všechny proměnné a všechny části, komponenty, struktury apod. modelu. Pomocí tohoto porozumění dokážeme odpovědět na otázku *Jak přesně vytváří natrénovaný model svou predikci?* Reálně je ovšem takováto úplná holistická globální interpretabilita téměř nedosažitelná, protože jakýkoli model, který má alespoň trochu větší počet vstupních proměnných a vah, je prakticky nemožné uložit v krátkodobé paměti člověka. Pokud jí ale dosáhneme, můžeme prohlásit, že model je plně interpretovatelný a to v nejstriktnějším slova smyslu[13].

Mnohem častější je tedy modulární globální interpretabilita. Ta odpovídá na otázku *Jak ovlivňují části modelu predikci?* Často to bývá to nejlepší, čeho můžeme u složitějších a komplexnějších modelů dosáhnout. Například naivní bayesovský klasifikátor spíše nebudeme schopni pochopit holisticky, nicméně pochopit jednotlivou váhu, tedy dosáhnout modulární (ale globální) interpretability bude poměrně snadné[12].

2.2.4.2 Lokální interpretabilita

Pokud se místo globálního, celkového, chování modelu zaměříme pouze na jednotlivou predikci, případně skupinu predikcí, a její vysvětlení (tedy na otázku

Proč model vytvořil pro danou instanci právě takovou predikci?), může se komplexní chování modelu stát mnohem snadněji pochopitelné a tedy interpretovatelné. Lokálně totiž může predikce na některých proměnných záviset lineárně, přestože globálně na nich má závislost mnohem komplexnější[12].

Například predikce hodnoty domu může záviset na jeho velikosti nelineárně. Pokud se ale bude zabývat pouze jedním domem, je přesto možné, že se predikce bude chovat vůči jeho velikosti lineárně. To se dá zjistit simulováním, jak by se predikovaná hodnota měnila se zvyšováním, příp. snižováním, velikosti domu, při zachování ostatních parametrů. Na konec je tedy možné, že lokální interpretabilita bude přesnější než globální.

2.2.5 Vlastnosti interpretability

Abychom měli možnost srovnání, porovnání a ohodnocení jednotlivých metod interpretability, je potřeba přiřadit jim, nebo jimi vytvářeným interpretacím, sadu vlastností. Problémem je, že ne všechny tyto vlastnosti umíme formálně a objektivně měřit. To zůstává jako jedna z nevyřešených otázek na poli interpretability[12].

2.2.5.1 Vlastnosti metod interpretability

- Expressive Power (vyjadřovací síla) – jazyk nebo struktura interpretace, kterou metoda vytváří. Může to být seznam IF-THEN pravidel, rozhodovací strom(y), váhy nebo dokonce interpretace v přirozeném jazyce apod.

- Translucency (průhlednost) – míra, nakolik metoda při tvorbě interpretace vhlíží přímo do modelu, např. na jeho parametry. Výhodou vyšší translucence může být lepší přesnost, protože metoda má k dispozici více dat, nevýhodou je ale přirozeně nižší portabilita.

- Portability (portabilita) – na jaké modely, nebo skupiny modelů, lze danou metodu použít. Portabilita je úzce provázána s translucencí, čím vyšší translucence, tím nižší je portabilita. Mezi nejpřenositelnější metody patří tzv. surrogate model, tj. metody, které vytvářejí „náhradní“ interpretovatelný model simulující chování interpretovaného modelu.

- Algorithmic Complexity (komplexita) – výpočetní složitost metody.

2.2.5.2 Vlastnosti interpretací

- Fidelity (věrnost) – *Jak přesně odhaduje interpretace predikci modelu?* Fidelity je asi nejdůležitější vlastností. Interpretace s nízkou fidelity je pro vysvětlení chování modelu nepoužitelná, jelikož mu vlastně neodpovídá. Mnohé metody ale garantují vysokou fidelity pouze lokálně, tedy

pro nějakou podmnožinu dat (např. local surrogate models), nebo dokonce pouze pro jednotlivou instanci (např. Shapleyho hodnoty). Pokud má interpretovaný model vysokou accuracy, pak interpretace s vysokou fidelity bude mít vysokou accuracy také.

- Accuracy (přesnost) – *Jak dobře dokáže interpretace predikovat nová, neznámá, data?* Vysoká přesnost je důležitá hlavně v případech, kdy chceme interpretaci používat *namísto* původního modelu. V ostatních případech je důležitá hlavně fidelity, která je ale s accuracy těsně provázaná.

- Consistency (konzistence) – *Jak moc se liší interpretace mezi podobnými modely trénovanými na stejných datech?* Tato vlastnost nemusí nutně vždy být úplně žádoucí, jelikož je možné, že modely se chovají podobně, přestože se nerozhodují na základě stejných proměnných. Pak by „správná“ interpretace neměla mít vysokou consistency[12].

- Stability (stabilita) – *Jak podobné jsou interpretace pro podobné instance?* Na rozdíl od consistency srovnává stabilita interpretace nad stejným modelem a tedy je vždy žádoucí. Zaručuje totiž, že drobná změna v instanci zásadně neovlivní interpretaci.

- Comprehensibility (pochopitelnost) – *Jak dobře dokáže člověk interpretaci pochopit?* Comprehensibility je jednou z nejdůležitějších, ale zároveň nejhůře měřitelnou vlastností. Interpretace může být ve všech ostatních ohledech sebelepší, ale pokud ji cílová audience nepochopí, pak je úplně zbytečná a stejně tak jsme mohli pouze ukázat model samotný.

- Certainty (jistota) – *Bere interpretace ohled na jistotu modelu?* Některé modely udávají predikce bez jakékoli confidence hodnoty – hodnoty vyjadřující míru „sebejistoty“, jistoty této predikce. Pokud interpretace takovou hodnotu přidává, může to být obrovská výhoda.

- Representativeness (reprezentativnost) – *Jakou část instancí interpretace pokrývá?* Interpretace může být globální, tedy pokrývající celý model, nebo klidně reprezentovat pouze jednu konkrétní predikci (např. Shapley Values), viz 2.2.4.

Kapitola 3

Metody interpretability modelů strojového učení

3.1 Permutation feature importance

Permutation feature importance je jednoduchá metoda pro určení důležitosti (importance) proměnné. Pro každou proměnnou permutujeme její hodnoty a pozorujeme, jak se změnila (pravděpodobně zhoršila) přesnost modelu. Předpokládáme, že pokud se po „přeházení“ hodnot proměnné výrazně sníží přesnost modelu, pak byla pro predikci důležitá. Pokud se ovšem téměř nezmění, pak daná proměnná neměla na predikci modelu vliv a tudíž není důležitá.

Tuto metodu poprvé představil Leo Breiman ve své práci o random forests[19], pro které je hojně používána. Tento přístup ale lze také generalizovat pro ostatní modely, jak ukázali Aaron Fisher, Cynthia Rudin a Francesca Dominici ve své práci[20].

3.1.1 Algoritmus pro výpočet permutation feature importance

Nechť $f(x)$ je natrénovaný model, X je vstupní matice proměnných, y je cílový vektor (požadované výstupy) a $L(y, f(x))$ je libovolná míra chyby, např. mean squared error.

1. Urči původní chybu modelu $e_{orig} = L(y, f(X))$.
2. Permutuj proměnnou, tj. vytvoř matici X_{perm}, kde zkoumaná proměnná má hodnoty permutované mezi instancemi.
3. Urči novou chybu modelu $e_{perm} = L(y, f(X_{perm}))$.
4. Urči hodnotu feature importance $FI = e_{perm} - e_{orig}$.

Pokud není míra $L(y, f(X))$ hodnotou chyby, tj. menší je lepší, ale naopak přesnost, tj. větší je lepší, pak se FI určuje jako $FI = e_{orig} - e_{perm}$[21]. Místo rozdílu lze také používat podíl $FI = e_{perm}/e_{orig}$, příp. opačný.

Pro lepší přesnost a stabilitu výsledků můžeme postup opakovat pro více permutací a výsledné FI průměrovat. Nicméně tato metoda je výpočetně náročná.

3.1.2 Trénovací nebo testovací množina?

Jako vstup do algoritmu pro výpočet permutation feature importance lze použít jak trénovací množinu, tj. množinu, na které se model učil, tak testovací, tj. jakákoli jiná množina vstupů. Nelze jednoznačně říct, který přístup je správný, protože obě možnosti nám přináší informaci, nicméně každý trochu jinou[12].

Pokud použijeme trénovací množinu, pak FI lépe vyjadřuje, jak se model chová, na základě jakých proměnných se opravdu rozhoduje. Dalo by se tedy říci, že tento přístup je lepší právě pro interpretabilitu ve smyslu zjišťování *na základě čeho vytváří model svou predikci.*

V opačném případě, tedy když použijeme množinu testovací, získáme lepší přehled o tom, které proměnné mají vliv na úspěšnost modelu v případě, kdy je vystaven novým, pro něj neznámým, datům. Poznáme, které proměnné ovlivňují jeho výkon v reálném případě a tudíž bychom se na ně měli zaměřit např. při kontrole kvality dat apod. Použití testovacích je také tou jedinou správnou možností, pokud chceme permutation feature importance použít pro feature selection (tj. redukci dimenzionality dat)[22].

3.1.3 Výhody permutation feature importance

- Jednoduchá a logická interpretace – feature importance vyjadřuje jak moc se zvýší chyba modelu, pokud daná proměnná ztratí svou informaci.
- Feature importance poskytuje rychle uchopitelnou globální informaci o modelu.

3.1.4 Nevýhody permutation feature importance

- Permutation feature importance je přímo svázána s chybou modelu a je tedy nutné znát správné hodnoty pro instance, se kterými pracujeme.
- Výsledky metody se mohou, kvůli používání náhodných permutací, zásadně lišit mezi jednotlivými běhy. Tuto nevýhodu můžeme, alespoň částečně, eliminovat použitím vyššího počtu náhodných permutací a jejich následným průměrováním. Metoda se pak ale stává velmi časově náročnou.

- V případě, kdy jsou proměnné korelované, je permutation feature importance nespolehlivá, jelikož hodnotí proměnné na základě výstupu modelu pro instance, které mohou být vysoce nepravděpodobné nebo dokonce nemožné. Stejným problémem trpí i partial dependance v následující části a příklad popisuji v části 3.4 o accumulated local effect grafech.

3.2 Partial dependence (PD)

Partial dependence graf je graf, vyjadřující funkční vztahy mezi jednotlivou (případně dvojicí, se zvyšujícím se počtem proměnných se zvyšuje počet dimenzí) proměnnou a predikcí modelu. PD graf může odhalit, zda je vztah mezi proměnnou a výstupní predikcí lineární, monotónní nebo nějak komplexnější. Např. může ukázat, že predikce ceny automobilu roste s výkonem motoru lineárně[14]. Představen byl poprvé J. H. Friedmanem v roce 2001 (1999)[15].

3.2.1 Definice

Matematicky je partial dependence funkce pro regresi definována jako:

$$f_S = \mathbb{E}_{\boldsymbol{x}_C}\left[f(\boldsymbol{x}_S, \boldsymbol{x}_C)\right] = \int f(\boldsymbol{x}_S, \boldsymbol{x}_C) dP(\boldsymbol{x}_C)$$

Kde $S \subset \{1, \ldots, p\}$, C je doplněk množiny S. $\boldsymbol{x}_S$ je vektor hodnot vstupních proměnných, pro které počítáme partial dependence a sjednocení $\boldsymbol{x}_S \cup \boldsymbol{x}_C$ tedy vytvoří celý prostor vstupů s kterými model $f(\boldsymbol{x})$ pracuje. Každá podmnožina proměnných S má vlastní partial dependence funkci f_S, která je dána průměrnou hodnotou f, když $\boldsymbol{x}_S$ je pevně daný a $\boldsymbol{x}_C$ se mění přes svoje marginální rozdělení $dP(\boldsymbol{x}_C)$. Tato funkce se aproximuje pomocí průměrů na trénovacích datech (tzv. metoda Monte Carlo) jako:

$$\hat{f}_S = \frac{1}{N} \sum_{i=1}^{N} \hat{f}(\boldsymbol{x}_S, \boldsymbol{x}_{Ci})$$

kde $\{\boldsymbol{x}_{C1}, \ldots, \boldsymbol{x}_{CN}\}$ označují hodnoty $\boldsymbol{x}_C$ v trénovacích datech[16].

3.2.2 Použití

Pro regresi se PD graf používá podle definice, pomocí aproximace přes průměry. To ovšem znamená, že předpokládáme nezávislost proměnných z množiny S na ostatních (z množiny C). Pokud budou korelované, výpočet zahrne i takové instance, které jsou velmi nepravděpodobné nebo dokonce nemožné[12].

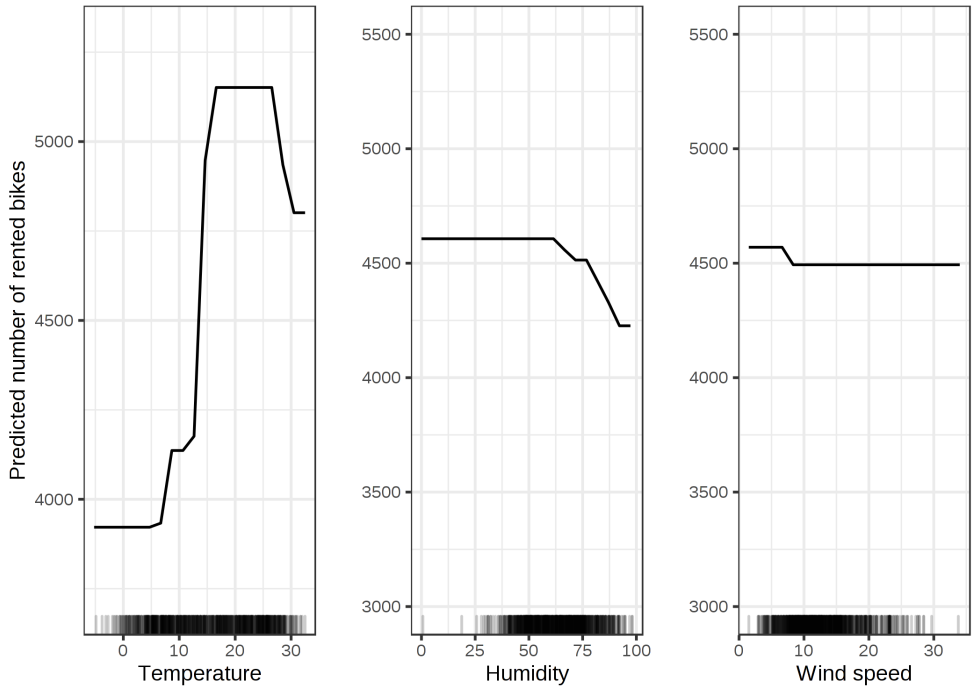
PD graf lze ale využít i pro klasifikaci. V tomto případě ukazuje pravděpodobnost zařazení do určité třídy, pro různé hodnoty v S. Pro vizualizaci je vhodné použít více grafů (každý pro jednu třídu), nebo více křivek v grafu.

Pokud je proměnná kategorická a ne numerická, výpočet partial dependence snadný. Jednoduše pro každou kategorii nahradíme všechny hodnoty

nacházející se v trénovacích datech postupně za jednotlivé možnosti. Například pokud proměnná vyjadřuje roční období, pak postupně všem instancím nastavíme hodnotu „zima“ a zprůměrujeme výstupy (predikce) modelu. To samé opakujeme pro ostatní roční období. Výstupem tedy budou 4 čísla a pro jejich vizualizaci bude nejjednodušší použít sloupcový graf, viz obrázek 3.2.

3.2.3 Ukázka použití

K ilustraci použití partial dependence grafů využiji ukázky z knihy Christopha Molnara[12]. Jako vstupní data slouží data o výpůjčkách jízdních kol společně s informacemi o počasí a sezóně od společnosti *Capital-Bikeshare*. V tomto příkladě byl natrénován random forest model pro předpověď počtu vypůjčených kol pro daný den a následně provedena analýza partial dependence.

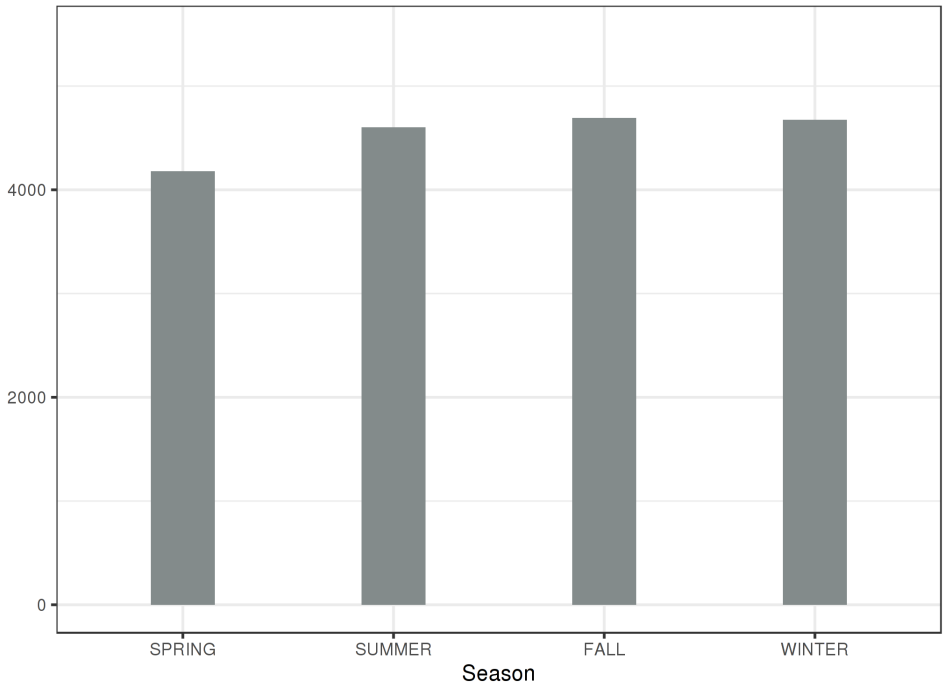


Obrázek 3.1: Ukázka PDP pro numerické proměnné[12]

Na obrázku 3.1 můžeme sledovat závislost predikce na teplotě (temperature), vlhkosti (humidity) a rychlosti větru (wind speed). Značky přímo u osy x vyjadřují rozdělení dat (tj. tam kde převládá černá barva máme v trénovacích datech dostatek instancí a naopak v bílých pásmech nedostatek).

Můžeme sledovat, že s rostoucí teplotou roste i počet výpůjček. To platí až cca k teplotě 26°C, kde začne počet vypůjčených kol klesat, což přibližně odpovídá očekávání. Na grafu týkajícím se rychlosti větru je zajímavé si

povšimnout, že byť by se dalo očekávat, že s rostoucí rychlostí větru bude klesat predikovaný počet výpůjček bez omezení, v grafu se pokles zastaví někde u hodnoty 25. To je pravděpodobně dáno nedostatečným počtem instancí a tudíž nepřesností modelu.



Obrázek 3.2: Ukázka PDP pro kategorickou proměnnou[12]

Obrázek 3.2 ilustruje, jak by mohl vypadat partial dependence graf pro kategorickou proměnnou „season“ (roční období). Z tohoto grafu můžeme usoudit, že roční období nemá až tak vysoký vliv na predikci tohoto modelu. Pozn. v grafu chybí popisek na ose y, nicméně je zde vyobrazen, podle očekávání, průměrný predikovaný počet výpůjček kol.

3.2.4 Výhody partial dependence

- Výpočet partial dependce, a tudíž i konstrukce PD grafů, je intuitivní. Hodnota partial dependence funkce v určité hodnotě proměnné je jednoduše průměrná predikce pro danou hodnotu.

- V případě, že proměnné, pro které je partial dependence počítána jsou *nekorelované* s ostatními, pak je interpretace naprosto jasná, graf zobrazuje jednoduše průměrnou predikci v daném bodě.

- Partial depndence je jednoduchá metoda na implementaci.

3.2.5 Nevýhody partial dependence

- Realisticky lze PD graf použít pouze pro jednotlivé proměnné nebo maximálně pro dvojice proměnných. To není dáno metodou, ale omezením vizualizace. Nejsme totiž schopni srozumitelně zobrazit více než třídimenzionální grafy.
- Je vhodné přidávat k partial dependence grafu také histogram rozložení nebo například značky podobně jako na obrázku 3.1. Pokud tuto informaci vynecháme, může být interpretace zavádějící.
- Složitější vztahy mohou zůstat skryty, jelikož PD graf ukazuje pouze průměrný marginální vliv. Pokud například u poloviny instancí bude zkoumaná proměnná přímo úměrná k predikci a u druhé nepřímo, pak bude křivka v PD grafu pravděpodobně monotónní, což by indikovalo, že proměnná nemá na predikci vliv. To ovšem není vůbec pravda. Tento problém řeší tzv. Individual Conditional Expectation (ICE), viz 3.3.
- Pravděpodobně největším nedostatkem PD grafů je ale již zmiňovaný předpoklad, že zkoumané proměnné jsou nekorelované s ostatními. Pokud jsou korelované, pak se do průměru započítávají i takové instance, které nemohou existovat (nebo mají jen velmi malou pravděpodobnost), což může poměrně zásadně ovlivnit výsledný graf. Jednou z alternativ řešících tento nedostatek je tzv. Accumulated Local Effects (ALE) graf, viz 3.4.

3.3 Individual conditional expectation (ICE)

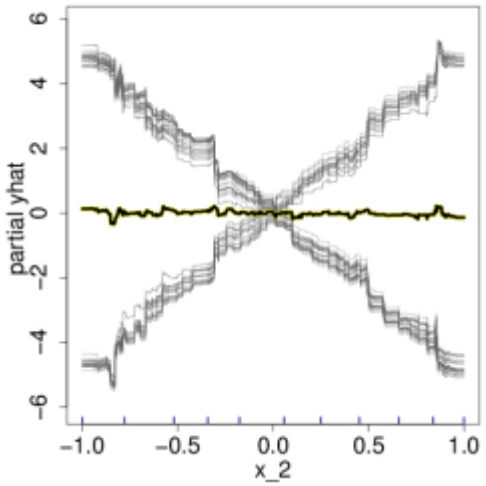
Jediným rozdílem mezi ICE a partial dependence grafy je, že ICE nevykresluje průměr, nýbrž jednu křivku pro každou instanci, čímž lze zabránit zaniknutí složitějších (heterogenních) vztahů. V podstatě můžeme tvrdit, že zprůměrováním všech křivek z ICE grafu vznikne PD graf. Obrázek 3.3 ilustruje rozdíl v chování obou grafů pro případ, kdy pro polovinu instancí predikce s rostoucí hodnotou proměnné lineárně roste a pro druhou polovinu klesá.

3.3.1 Definice

Pro každou instanci z $\{(x_S^{(i)}, x_C^{(i)})\}_{i=1}^N$, kde N je celkový počet instancí a $x_S^{(i)} \cup x_C^{(i)}$ tvoří jednu kompletní instanci i, je vykreslena křivka $\hat{f}_S^{(i)} = \hat{f}(x_S^{(i)}, x_C^{(i)})$ pro měnící se hodnoty $x_S^{(i)}$ a zafixované $x_C^{(i)}$.

3.3.2 Centered ICE

Nevýhodou ICE grafů je, že může být těžké rozlišit, zda a jak se jednotlivé křivky liší. Je to proto, že každá křivka začíná v jiném bodě – v jiné pre-



Obrázek 3.3: Porovnání PD a ICE[16]. Výraznější křivka pohybující se kolem 0 je křivka partial dependence, zbytek je ICE graf.

dikci. Jednoduché řešení přináší tzv. centered ICE (c-ICE), graf, který posune některý, většinou nejnižší bod na ose x, tj. bod s nejnižší hodnotou zkoumané proměnné, do 0:

$$\hat{f}_{cent}^{(i)} = \hat{f}^{(i)} - \mathbf{1}\hat{f}(x^a, x_C^{(i)})$$

Kde $\mathbf{1}$ je vektor 1 se správným počtem dimenzí (reálně většinou 1) a x^a označuje tzv. anchor point – bod ve kterém bude křivka v 0. Pokud bude x^a nejnižší hodnota x_S, pak budou křivky začínat v 0[16], díky tomu bude graf čitelnější.

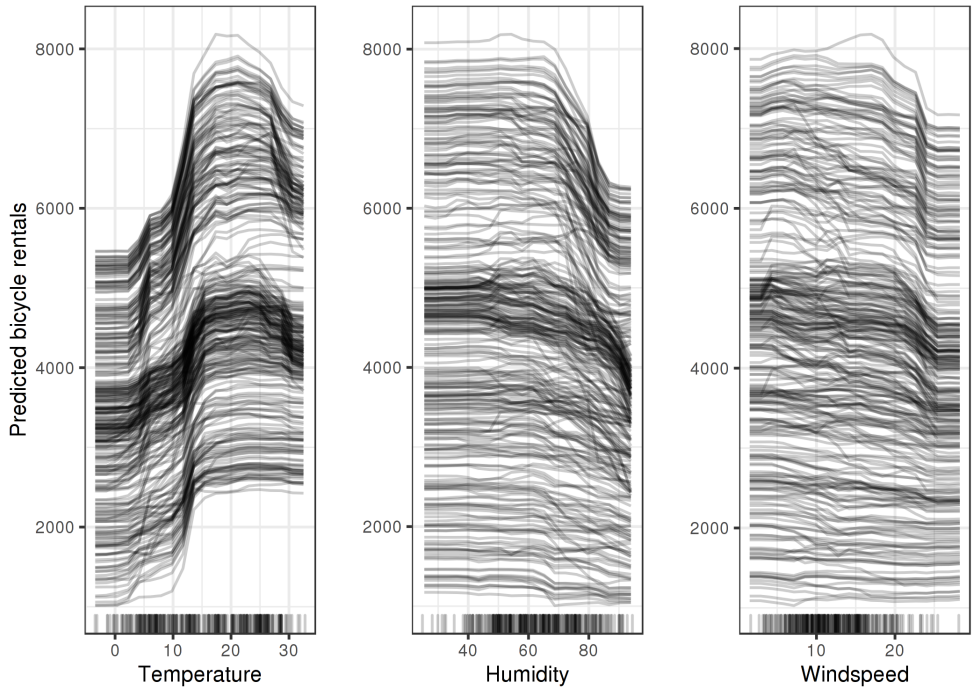
3.3.3 Derivative ICE

Další možností jak snadněji rozpoznat heterogenitu v datech je tzv. derivative ICE (d-ICE) graf, který zobrazuje derivaci $\hat{f}$ podle x_S. Jinými slovy zobrazuje, kdy a jakým směrem dochází ke změně predikce. Pomocí d-ICE grafu lze snadno rozpoznat, kde dochází ke komplexnějším interakcím mezi x_S a x_C. Pokud by totiž žádné takové interakce nenastávaly, byly by všechny křivky v d-ICE grafu stejné. Nicméně to samé platí i pro c-ICE graf.

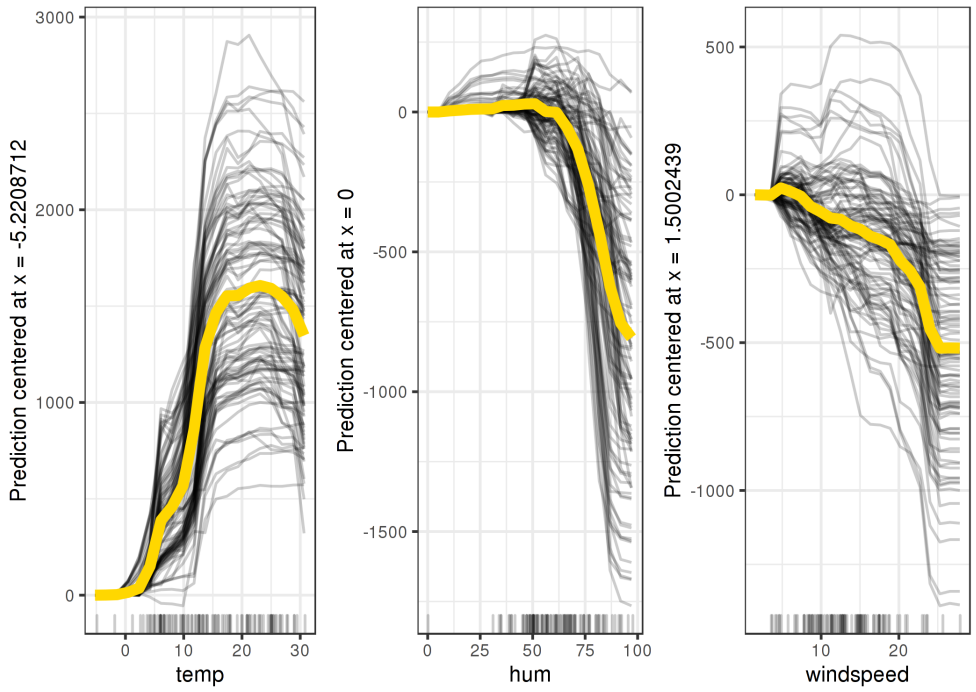
3.3.4 Ukázka použití

Použití ICE grafů ilustruji na stejném příkladě jako v sekci 3.2.3, tedy na modelu předpovídajícím počet výpůjček jízdních kol.

Obrázek 3.4 ukazuje ICE grafy, stejně jako PD graf 3.1, který by vznikl „zprůměrováním“ zobrazených křivek, pro 3 proměnné – teplotu, vlhkost a sílu větru. Tento ICE graf nám oproti PD grafu poskytuje další informaci.



Obrázek 3.4: Ukázka ICE grafů[12]



Obrázek 3.5: Ukázka c-ICE grafů[12]

To, že většina křivek má podobný průběh naznačuje, že PD graf 3.1 je dostatečně vypovídající. Je totiž vidět, že zprůměrováním nedošlo k žádnému významnému zkreslení jako například na obrázku 3.3.

Centered ICE grafy na obrázku 3.5 ještě vylepšují pohled na ta samá data jejich vycentrováním. Žlutá čára značí průměr, tedy vlastně centrovaný PD graf. Je vidět, že přesto, že u tohoto příkladu je PD graf poměrně přesný, zůstaly některé vztahy poněkud skryty. Například můžeme pozorovat, že rostoucí rychlost větru pro některé instance predikci nejprve zvyšuje a teprve později ji snižuje a pro některé rovnou snižuje. Proto můžeme říct, že rychlost větru nebude sama o sobě dobrým prediktorem a v modelu existují nějaké složitější vztahy, které PD graf skrývá.

3.3.5 Výhody ICE grafů

- ICE grafy jsou pravděpodobně ještě intuitivnější než PD grafy.
- Oproti PD grafům dokáže ICE zobrazit i složitější, heterogenní vztahy.
- Podobně jako PD je i ICE jednoduchá metoda na implementaci (pozn. to neplatí pro d-ICE, jejíž výpočet je *dlouhý a poměrně nepraktický*[12]).

3.3.6 Nevýhody ICE grafů

- ICE je prakticky nemožné použít pro více jak jednu proměnnou. Už pro dvě proměnné bude výsledkem několik prolínajících se ploch, což bude naprosto nepřehledné a graf tedy nebude mít žádnou vypovídací hodnotu.
- I pro jednu proměnnou se ovšem může stát, že v grafu bude jednoduše příliš mnoho křivek a ten se tudíž stane nepřehledným. To se dá řešit např. samplingem (vzorkováním) nebo tím, že křivky budou částečně průhledné.
- Stejně jako u PD grafů platí, že zkoumaná proměnná by měla být nekorelovaná s ostatními. Viz 3.2.5.

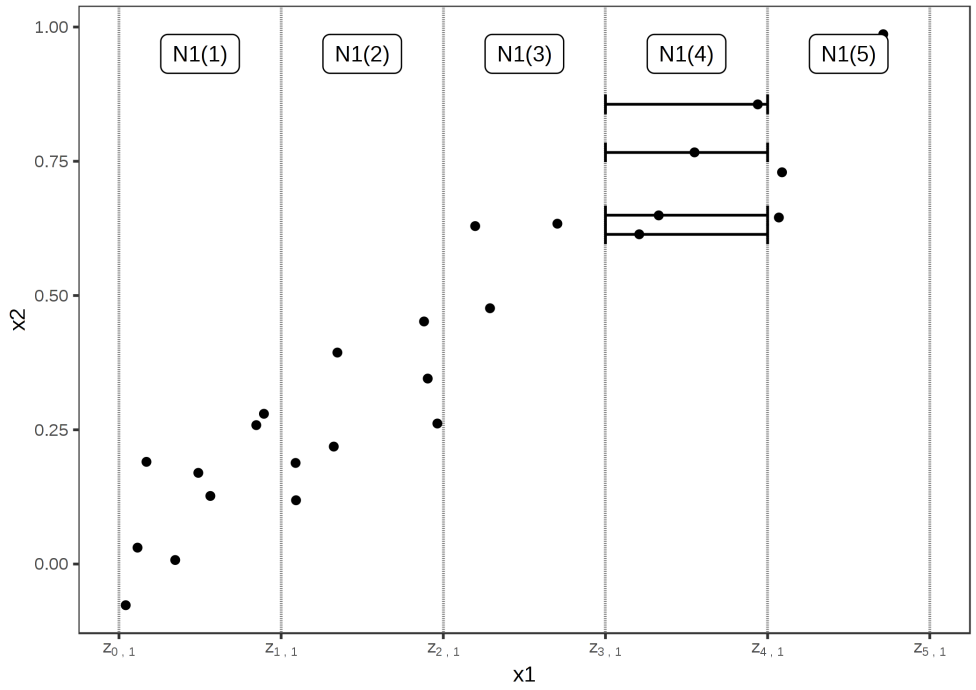
3.4 Accumulated local effects (ALE)

Accumulated local effects grafy popisují, podobně jako PD grafy, jak proměnná ovlivňuje průměrnou predikci modelu. Na rozdíl od PD grafů se ALE umí vypořádat i se situacemi, kdy je zkoumaná proměnná korelovaná s některou jinou proměnnou.

Proč vlastně není partial dependence vhodná pro proměnné, které jsou korelované? Protože do průměru zahrnuje i takové instance, které nejsou pravděpodobné, nebo jsou dokonce *nemožné*. Například budeme-li konstruovat PD

graf pro obytnou plochu u modelu predikujícího cenu domu, budeme postupovat tak, že osu x (obytná plocha) rozdělíme na diskrétní body, pro každý z bodů nahradíme obytnou plochu ve všech instancích příslušnou hodnotou a predikce modelu pro vzniklé instance zprůměrujeme. Tím vytvoříme body v grafu, které proložíme křivkou. To znamená, že do průměru v bodě 30m^2 se budou započítávat i hypotetické domy s 10 pokoji a obytnou plochou pouze 30m^2.

ALE tento nedostatek řeší tak, že pro každý takový bod počítá rozdíly v predikci, pokud by se obytná plocha měnila. Pro zmíněný bod 30m^2 použije ALE všechny instance, které mají obytnou plochu přibližně 30m^2 a zprůměruje rozdíly predikcí při nahrazení obytné plochy za 31m^2 mínus predikce pro 29m^2. Tím dostane čistě efekt obytné plochy odstíněný od korelovaných proměnných.



Obrázek 3.6: Konstrukce ALE grafu[12]

Obrázek 3.6 ilustruje konstrukci ALE grafu pro proměnnou x1, korelovanou s proměnnou x2. Obor proměnné se nejprve rozdělí na intervaly (svislé čáry). Následně se pro každý interval vypočítá rozdíl v predikcích pro instance do něj spadajících při nahrazení hodnoty zkoumané proměnné za horní mez intervalu a za dolní mez. Tyto rozdíly se nakonec akumulují a vycentrují, čímž vznikne ALE graf.

3.4.1 Definice

Rovnice pro ALE[12]

$$\begin{aligned}\hat{f}_{x_S,ALE}(x_S) &= \int_{z_{0,1}}^{x_S} E_{X_C|X_S}\left[\hat{f}^S(X_s, X_c)|X_S = z_S\right] dz_S - \text{cons} \\ &= \int_{z_{0,1}}^{x_S}\int_{x_C} \hat{f}^S(z_s, x_c)\mathbb{P}(x_C|z_S)dx_C dz_S - \text{cons}\end{aligned}$$

Stejně jako v definici partial dependence 3.2.1 označují X_S zkoumanou podmnožinu vstupních proměnných a X_C zbylé vstupní proměnné. Na rozdíl PD se ale neprůměrují hodnoty predikce, nýbrž jejich rozdíly, definované jako gradient

$$\hat{f}^S(x_s, x_c) = \frac{\delta \hat{f}(x_S, x_C)}{\delta x_S}$$

Takový gradient ovšem neexistuje pro všechny modely. Jedním z modelů, které tento gradient nemají, je například random forest. To není problém, protože tato funkce se ve skutečnosti pouze aproximuje a to bez gradientů, pomocí intervalů.

Na konec se od výsledku ještě odčítá konstanta *cons*, která zajistí vycentrování ALE grafu tak, aby průměrná hodnota byla 0 a tedy aby průměrný vliv (effect) zkoumané proměnné byl 0.

3.4.2 Aproximace ALE

Nevycentrovaně vypadá ALE následovně:

$$\hat{\tilde{f}}_{j,ALE}(x) = \sum_{k=1}^{k_j(x)} \frac{1}{n_j(k)} \sum_{i:x_j^{(i)} \in N_j(k)} \left[f(z_{k,j}, x_{\setminus j}^{(i)}) - f(z_{k-1,j}, x_{\setminus j}^{(i)})\right]$$

Poslední suma sčítá vliv proměnné na všechny instance uvnitř intervalu (označeném jako $N_j(k)$ – okolí). Ten se poté zprůměruje zlomkem $1/n_j(k)$, čímž dostaneme jakýsi lokální vliv (local effect). Tyto lokální vlivy nakonec akumuluje (sčítá) poslední suma. ALE proměnné pro hodnotu ležící např. ve třetím intervalu je tedy součet vlivů z prvního, druhého a třetího intervalu.

Tato hodnota se pak ještě centruje tak, aby průměrný vliv byl 0:

$$\hat{f}_{j,ALE}(x) = \hat{\tilde{f}}_{j,ALE}(x) - \frac{1}{n}\sum_{i=1}^{n} \hat{\tilde{f}}_{j,ALE}(x_j^{(i)})$$

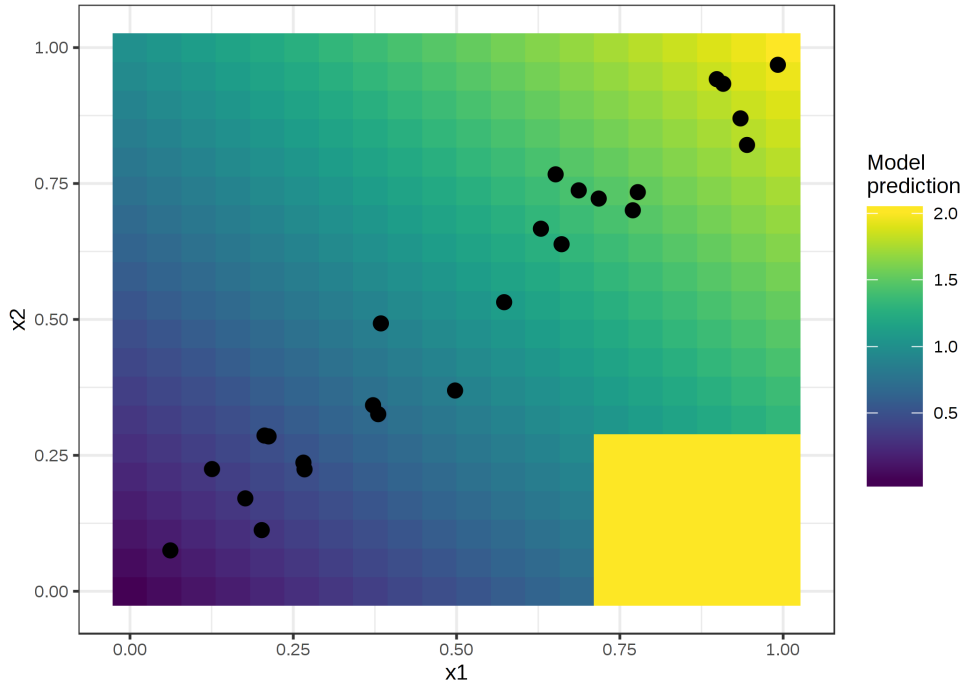
Podobně jako PD se i ALE dá vyjádřit pro libovolný počet proměnných, nicméně opět jsme při interpretaci omezeni možnostmi zobrazení, které prakticky neumožňují více jak 2 proměnné, proto se touto variantou nebudu zabývat. Přesnější matematické definice a ALE pro více proměnných je popsáno v [17].

3.4.3 Interpretace ALE

ALE hodnota se dá interpretovat jako vliv, jaký bude mít konkrétní hodnota proměnné v porovnání s průměrnou predikcí. Například pokud bude ALE $\hat{f}_{j,ALE}(3) = 1$, pak predikce v případě, že proměnná $x_j = 3$, je o 1 vyšší než průměrná predikce.

3.4.4 ALE pro kategorické proměnné

Accumulated local effects potřebuje, aby proměnná měla nějaké uspořádání, protože tato metoda sčítá výsledky v jednom směru. Jenomže kategorické proměnné nemusí mít, a mnohdy nemají, přirozené uspořádání. I pro takové proměnné lze ALE napočítat, jen je potřeba nějaké uspořádání jim vytvořit. Často používanou možností je uspořádat kategorie podle jejich podobnosti vzhledem k ostatním proměnným. Vzdálenost mezi dvěma kategoriemi je suma vzdáleností každé z proměnných. Vzdálenost proměnných je pak počítána jako tzv. Kolmogorov-Smirnov (někdy jen Kolmogorov) distance[18] pro numerické proměnné a nebo pomocí tabulek relativní četnosti pro kategorické proměnné.

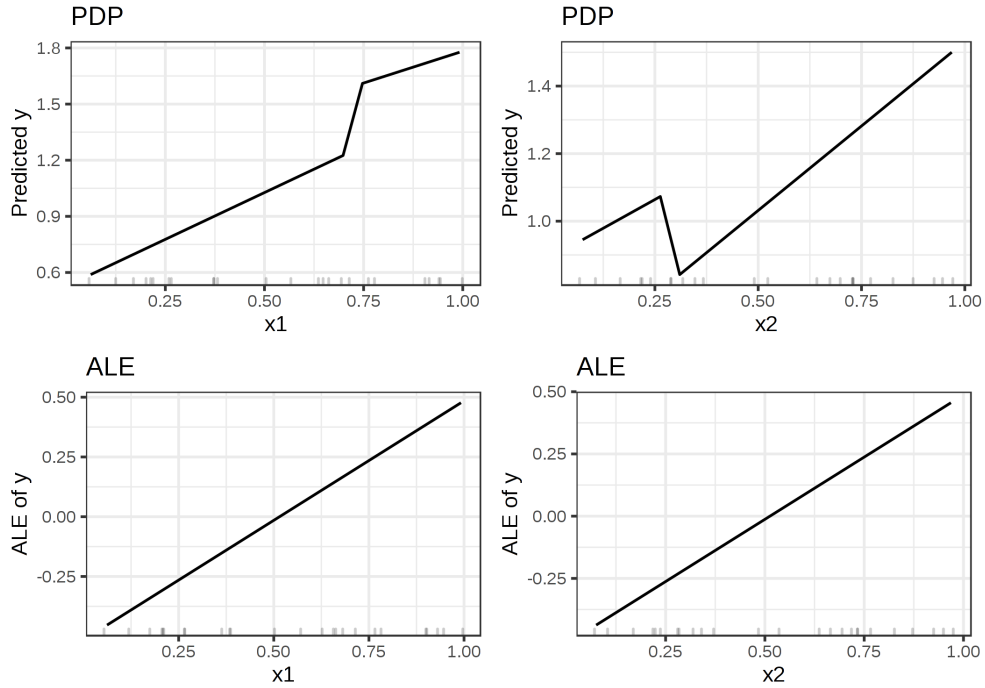


Obrázek 3.7: Model, pro který PD selhává. Černé tečky jsou datové body v trénovacím datasetu a barva pozadí značí predikci modelu pro danou kombinaci hodnot $x1$ a $x2$[12].

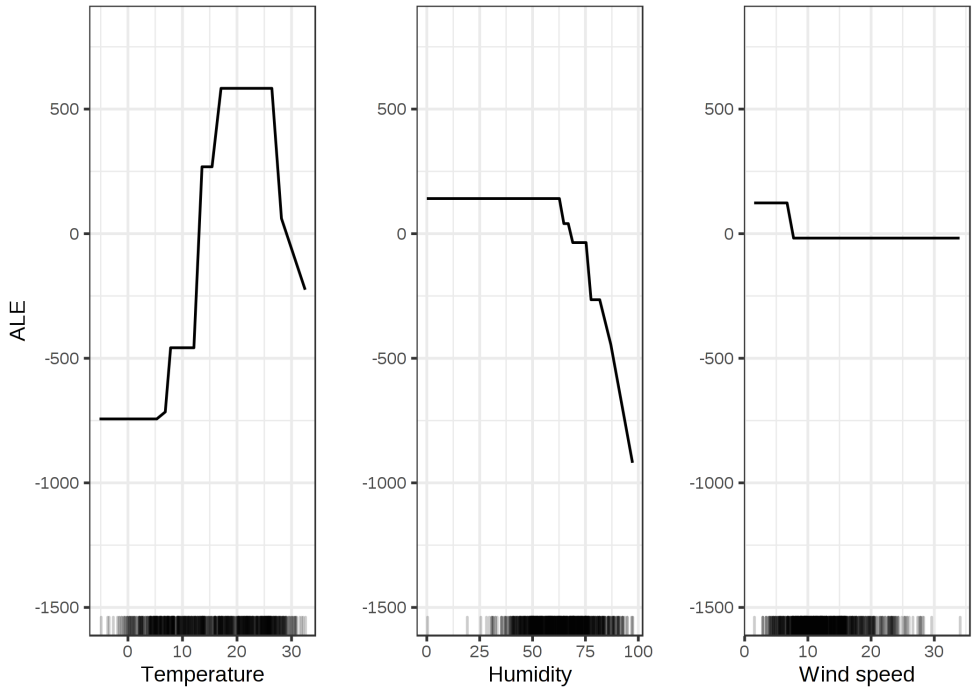
3.4.5 Ukázka použití

Na prvním příkladu bude jasně vidět, proč je v případě korelovanosti proměnných lepší používat ALE oproti PD grafům. Uvažujme model, který predikuje výstup na základě dvou silně korelovaných proměnných, ale pro kombinace, které se v datech nevyskytují se chová zvláštně, viz 3.7. To nemá žádný vliv na přesnost modelu, jelikož tyto kombinace jsou extrémně nepravděpodobné, takže se ve skutečnosti nevyskytují. Ačkoli je tento příklad vytvořen uměle, technicky vzato může k takovým situacím docházet i v reálných případech, protože model není při učení za takové chování nijak penalizován.

Výstupy obou metod jsou na obrázku 3.8. ALE identifikoval lineární závislost predikce na jednotlivých proměnných a ignoroval při tom oblast, ve které se model nechová podle očekávání. Naopak PD graf je zásadně ovlivněn zmiňovanou oblastí. Může vyvstat otázka, zda je správné tuto oblast ignorovat nebo ne. Rozhodně může být zajímavé zaznamenat, že v modelu dochází k něčemu neočekávanému. Na druhou stranu se jedná, nebo by se alespoň mělo jednat, o instance, které jsou buď nepravděpodobné nebo dokonce i nemožné. V případě, že by tomu tak nebylo, je vhodné zvážit, zda je trénovací dataset dostatečně reprezentativní. V každém případě je potřeba rozhodnutí, zda se chceme takovými případy zabývat nebo ne, učinit vědomě.



Obrázek 3.8: Porovnání ALE a PD grafů pro model ilustrovaný obrázkem 3.7[12].



Obrázek 3.9: Ukázka ALE grafů[12].

Obrázek 3.9 ukazuje ALE grafy pro již několikrát zmíněný dataset z půjčovny kol. V porovnání s PD grafy z kapitoly o partial dependence (obrázek 3.1) můžeme pozorovat, že pokles predikce s vysokou teplotou a vlhkostí je podle ALE strmější. Jinak jsou ale výstupy obou metod velmi podobné.

3.4.6 Výhody ALE

- ALE lze použít i když jsou proměnné korelované.
- Výpočetní složitost ALE grafů je pouze $O(n)$, tudíž konstrukce ALE grafu je velmi rychlá.

3.4.7 Nevýhody ALE

- Určit počet (a tím i velikost) intervalů může být problematické. Pokud bude intervalů příliš mnoho může křivka ALE grafu obsahovat mnoho malých výkyvů. Pokud jich naopak bude málo, nebude ALE dostatečně přesné.
- Samotná implementace je mnohem složitější než u partial dependence nebo ICE.

3.5 Global surrogate model

Global surrogate (zástupný) model je nějaký interpretovatelný model, který natrénujeme tak, aby předpovídal výstupy black-box modelu, který chceme interpretovat. Surrogate model může být jakýkoli model, který umíme snadno interpretovat, jako třeba lineárně regresní model nebo rozhodovací strom.

3.5.1 Koeficient determinace

Abychom mohli rozhodnout, zda natrénovaný surrogate model dobře aproximuje zkoumaný model, potřebujeme nějakou míru chyby nebo přesnosti. Takovou mírou může být tzv. koeficient determinace (R-squared measure):

$$R^2 = 1 - \frac{SSE}{SST} = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_*^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{n}(\hat{y}^{(i)} - \bar{\hat{y}})^2}$$

kde $\hat{y}_*^{(i)}$ je predikce surrogate modelu pro i-tou instanci, $\hat{y}^{(i)}$ je predikce zkoumaného modelu pro i-tou instanci a $\bar{\hat{y}}$ je střední hodnota predikcí zkoumaného modelu. SSE značí „sum of square errors“, tj. suma čtverců chyb (residuí) – vlastně variabilita nepřesností aproximace, SST je „sum of squares total“, tj. celková suma čtverců – variabilita predikcí zkoumaného modelu[12].

Koeficient determinace můžeme chápat jako podíl variability predikcí zkoumaného modelu, který je vysvětlen (pokryt) surrogate modelem. Pohybuje se v intervalu $\langle 0, 1\rangle$. Pokud se hodnota blíží 1, pak je SSE malá a tedy surrogate model velmi dobře aproximuje zkoumaný model. V takovém případě lze uvažovat i o nahrazení původního modelu tímto novým interpretovatelným modelem, jelikož má velmi podobné chování a výhodu interpretovatelnosti. Naopak v případě, že se R^2 blíží 0, neplní surrogate model svoji funkci, jelikož aproximuje zkoumaný model velmi nepřesně a pravděpodobně nebude interpretace tohoto modelu aplikovatelná na model původní.

3.5.2 Výhody global surrogate model

- Flexibilita – můžeme použít jakýkoli model, který dokážeme interpretovat.
- Pomocí koeficientu determinace můžeme určit jak přesná („dobrá“) je interpretace, kterou surrogate model poskytuje.

3.5.3 Nevýhody global surrogate model

- Surrogate model se může chovat velmi podobně pro určitou podmnožinu dat, ale pro jinou část být naprosto odlišný. Může se tedy stát, že interpretace nebude přesná pro všechny instance.

- Každý model, který použijeme jako surrogate má své vlastní výhody a nevýhody.

3.6 LIME

Local interpretable model-agnostic explanations (LIME) je algoritmus, který dokáže vytvořit vysvětlení, pro predikci jakéhokoli modelu tak, že vytvoří zástupný model, který jej *lokálně* aproximuje. Jedná se tedy o metodu pro vysvětlení jednotlivých predikcí.

3.6.1 Definice

Formálně definujeme vysvětlení (explanation) ξ, které je výstupem LIME jako model $g \in G$, kde G je množina potenciálně interpretovatelných modelů, např. linárně regresní modely nebo rozhodovací stromy. Protože ale ne všechny modely z $g \in G$ musí být interpretovatelné zavádíme ještě $\Omega(g)$ – míru komplexity vysvětlení $g \in G$. LIME tedy produkuje následující vysvětlení:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

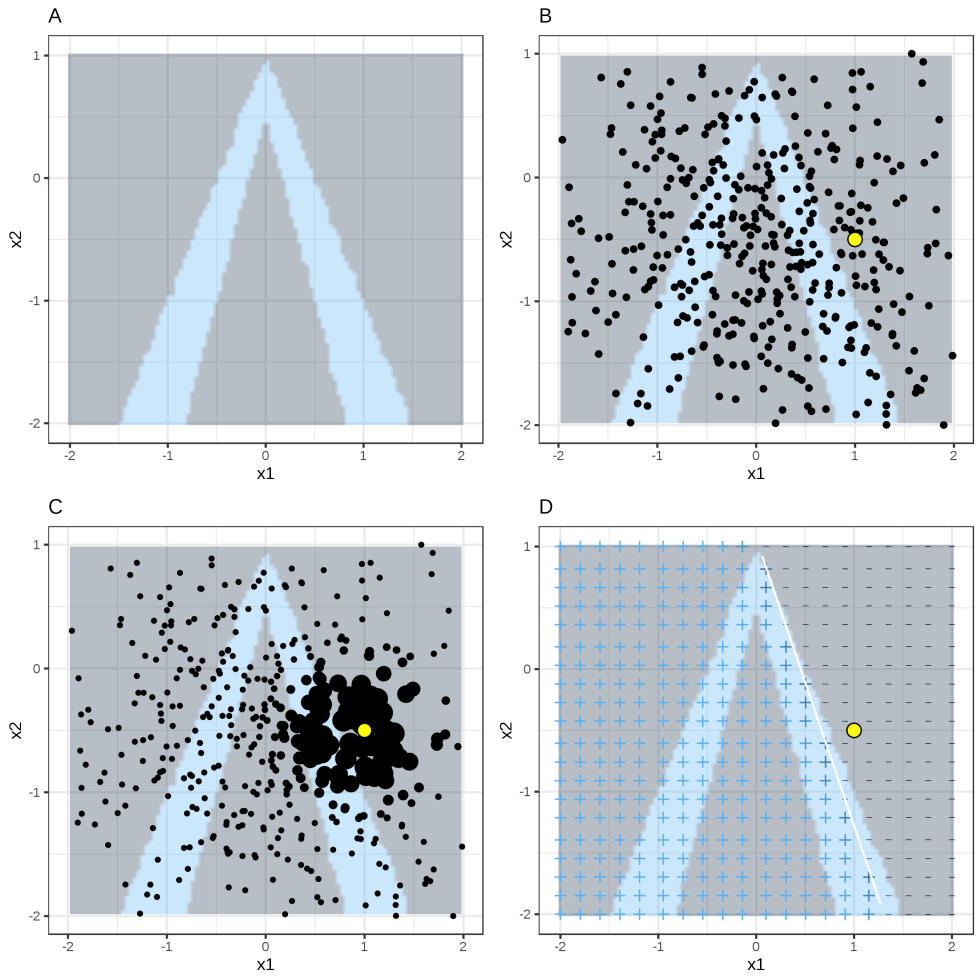
$\mathcal{L}(f, g, \pi_x)$ je míra chyby, např. RMSE, modelu g vůči zkoumanému modelu f. Dále ještě do $\mathcal{L}()$ vstupuje proximity measure (míra blízkosti) π_x, která definuje velikost okolí instance x, které pro vysvětlení uvažujeme[23].

V praxi LIME optimalizuje jenom $\mathcal{L}(f, g, \pi_x)$. Komplexitu $\Omega(g)$ si uživatel nastavuje sám, například pomocí omezení maximálního počtu proměnných, které lineárně regresní model může použít.

3.6.2 Algoritmus

Cílem je tedy natrénovat model, tak abychom minimalizovali ztrátovou funkci $\mathcal{L}(f, g, \pi_x)$, bez znalosti f. Pokud bychom o f něco předpokládali, pak už by se nejednalo o agnostickou metodu. $\mathcal{L}(f, g, \pi_x)$ aproximujeme pomocí samplingu (náhodného výběru) z okolí x, které necháme modelem f ohodnotit a přiřadíme jim váhu pomocí π_x. Na tomto výběru poté natrénujeme interpretovatelný model. Sampling okolí x provádíme pomocí tzv. random uniform sampling, tedy tak, že odebíráme náhodné nenulové množství proměnných (features) instance x a těmto „poničeným“ instancím dogenerováváme odebrané proměnné na základě normálního rozdělení[23]. Na základě jedné vysvětlované instance tedy vytvoříme celý trénovací dataset pro interpretovatelný model.

Obrázek 3.10 ilustruje, jak LIME pracuje. Obrázek A ukazuje predikce nějakého klasifikátoru (v tomto případě se jedná o random forest), který výtváří predikce na základě proměnných $x1$ a $x2$. Na obrázku B jsme zvolili instanci, kterou chceme vysvětlit – vyznačena žlutě, a provedli random uniform sampling, čímž jsme získali trénovací dataset (zatím bez vah). Na obrázku C



Obrázek 3.10: Ukázka procesu LIME[12].

jsme k instancím již přidali váhu pomocí π_x, což je naznačeno velikostí tečky reprezentující instanci. A konečně na obrázku D je výsledný natrénovaný interpretovatelný model (lineárně regresní model), symboly plus a mínus značí jeho predikci. Můžeme pozorovat, že lokálně tento model poměrně věrně odpovídá původnímu modelu. Nicméně z globálního hlediska zdaleka není tak odpovídající. V tomto případě by lineární model ani nemohl být příliš vypovídající globálně, což znovu ilustruje nevýhody global surrogate model.

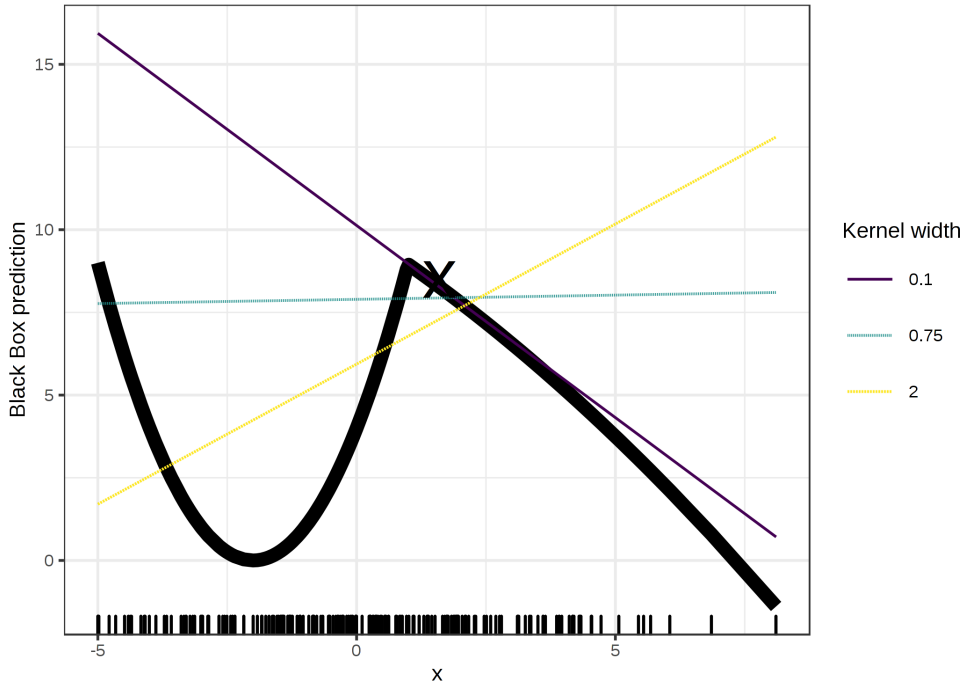
3.6.3 Proximity measure

Jako proximity measure π_x se používá tzv. exponential smoothing kernel $\pi_x(z) = \exp(-D(x,z)^2)/\sigma^2$, kde $D(x,z)$ je funkce, která vrátí vzdálenost mezi x a z, jako σ se označuje tzv. kernel width. Ztrátovoá funkci $\mathcal{L}$ je pak *locally-*

aware square loss:

$$\mathcal{L}(f, g, \pi_x) = \sum_{z,z' \in \mathcal{Z}} \pi_x(z)(f(z) - g(z'))^2$$

Nicméně právě *kernel width* σ je problémová část LIME. Nejsme totiž jednoduše schopni určit, jak velká má tato hodnota být. Ve většině implementací (např. Python implementace LIME) se používá hodnota $0,75$, respektive $0,75 \times \sqrt{počet\ sloupců}$, pokud uživatel nezadá vlastní. Kernel width v podstatě říká, pro jak velké okolí instance x se má interpretovatelný model snažit být přesný. Pokud bude okolí příliš velké, pak se v podstatě vracíme k global surrogate model a pokud bude naopak příliš malé, nemusí být interpretace vůbec vypovídající, protože surrogate model bude mít za úkol se s původním modelem shodovat jen ve velmi malé oblasti. Vliv kernel width ilustruje obrázek 3.11.



Obrázek 3.11: Ukázka vysvětlení predikce modelu závislého na 1 proměnné pomocí LIME pro instanci $x = 1,6$ a různé kernel width[12].

3.6.4 Výhody LIME

- LIME je velmi univerzální metoda, lze ji použít pro jakýkoli model, ale také pro jakákoli data – nejen pro standardní tabulková data, ale i pro obrázky nebo text.

- Fidelity se pro vysvětlení generované pomocí LIME dobře určuje a dává nám dobrý odhad, jak je vysvětlení použitelné (přesné).
- Díky možnosti využít v podstatě jakýkoli model jako surrogate může být výsledná interpretace taková, jakou ji chceme. Například pokud použijeme Lasso trees, bude vysvětlení krátké a snadno pochopitelné pro člověka. Nebude však nutně *plně* vysvětlovat predikci.

3.6.5 Nevýhody LIME

- Největší nevýhodou LIME je bezesporu zmiňovaná kernel width, popsaná v sekci 3.6.3 o proximity measure.
- V dosavadní implementaci LIME není sampling úplně dokonalý. Datové body jsou tvořeny náhodně, podle normálního rozdělení, která ignoruje korelaci a vztahy v datech. Může se tedy jednat i o body, které jsou nepravděpodobné nebo nemožné.
- LIME má malou stabilitu. Interpretace pro podobné nebo i stejné body se tedy mohou zásadně lišit.

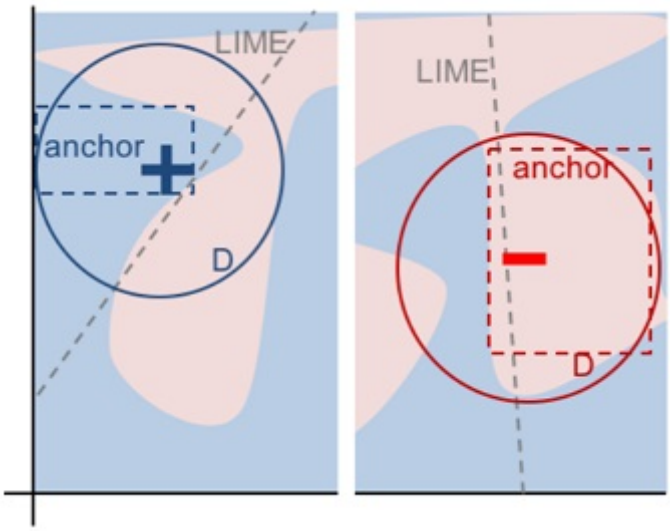
3.7 Anchors

V roce 2018 představili autoři LIME, Marco Tulio Ribeiro, Sameer Singh a Carlos Guestrin, nový algoritmus, který vysvětluje predikce pomocí pravidel zvaných *anchors* (kotvy), představujících lokální dostačující podmínky pro danou predikci[24]. Podobně jako LIME prohledává i Anchors okolí vysvětlované instance. Na rozdíl od LIME ale netrénuje zástupný model, nýbrž vytváří *IF-THEN* pravidlo, které bude s dostatečnou, uživatelem předem definovanou, pravděpodobností (precision) platné pro všechny instance v datech. Takové pravidlo je pak použitelné pro vysvětlení instancí, splňujících všechny predikáty s pravděpodobností rovnou precision.

```
IF SEX = female
AND Class = first
THEN PREDICT Survived = true
WITH PRECISION 97%
AND COVERAGE 15%
```

Ukázka 3.12: Ukázka IF-THEN pravidla, které by mohlo být výstupem Anchors metody pro model předpovídající přežití pasažéra na Titanicu.

Obrázek 3.13 ukazuje vizualizaci srovnání LIME a Anchors pro 2 instance. Oba algoritmy vysvětlují predikce modelu s dvěma výstupy (+ nebo −) za použití okolí $\mathcal{D}$. LIME natrénuje lineární model, který aproximuje původní model, ale neposkytuje informaci o tom, jak je aproximace přesná, a pro



Obrázek 3.13: Srovnání LIME a Anchors pro 2 různé instance[24].

které, pokud vůbec pro nějaké, další instance je toto vysvětlení použitelné. Naproti tomu Anchors má, z definice, jasně dané „pole působnosti“, podprostor, ve kterém lze vysvětlení využít (s pravděpodobností danou precision).

3.7.1 Definice

Pravidlo (Anchor) A a jeho precision $prec(A)$ se definuje jako

$$prec(A) = \mathbb{E}_{\mathcal{D}_x(z|A)}[1_{f(x)=f(z)}] \geq \tau, A(x) = 1$$

kde x je vysvětlovaná instance, A je množina predikátů tvořících pravidlo (např. `sex = female`), takových že $A(x) = 1$, pokud všechny predikáty pravidla A jsou platné v instanci x; f vysvětlovaný model; $\mathcal{D}_x(z|A)$ značí „sousedy“, tj. okolí instance x, pro které platí A a $0 \leq \tau \leq 1$ je *precision threshold*, tj. minimální precision, které musí pravidlo A dosáhnout[24].

Precision τ je vlastně míra fidelity (věrnosti), tedy míra toho, jak přesně odpovídá vysvětlení, zde pravidlo A, výstupům modelu. Samozřejmě se snažíme dosáhnout co největší precision, nicméně zvětšováním precision se snižuje coverage – pravděpodobnost, že pravidlo A bude aplikovatelné i na sousedy instance ($cov(A) = \mathbb{E}_{\mathcal{D}_{(z)}[A(z)]}$).

Dosáhnout $\tau = 1$ můžeme vždy tak, že vytvoříme takové pravidlo A, které obsahuje predikát pro každou proměnnou. Tím se ale pravidlo stane v podstatě zbytečné, protože nepřináší žádnou novou informaci. Vlastně jen opakuje, že pro danou konfiguraci proměnných vrací model tuto predikci. Dokonalou coverage bude zase mít pravidlo A bez predikátů. Jenomže takové pravidlo, např. `PREDICT Survived = true`, bude mít nejen nízkou precision

(pravděpodobně, například u modelu predikujícího přežití cesty autem může mít i takové pravidlo poměrně velkou precision), ale hlavně nám o fungování modelu neříká vůbec nic. Vlastně existuje možnost, že říká, ale pokud tomu tak je, je náš model poněkud zbytečný. Přirozeně se tedy metoda snaží najít kompromis mezi precision a coverage.

3.7.2 Algoritmus hledání pravidel

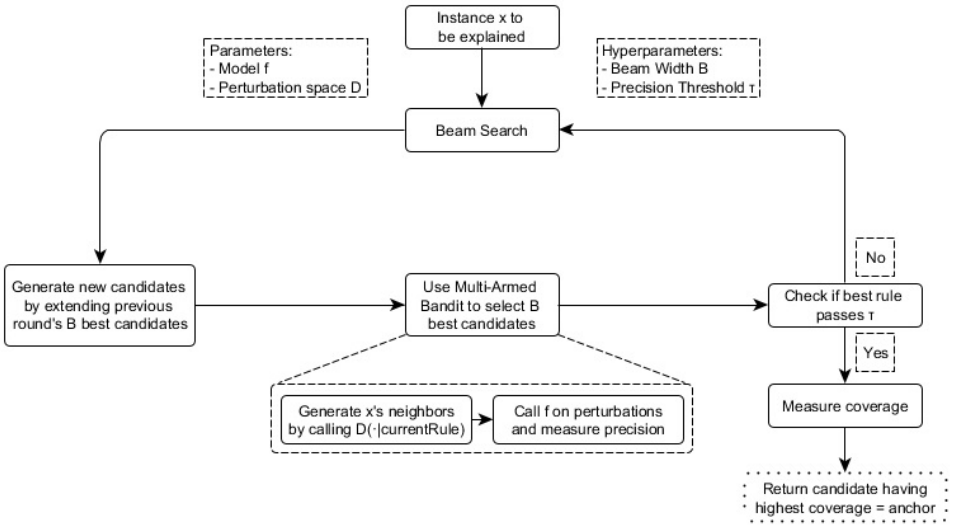
Protože není reálně možné konstruovat pravidla z definice, jelikož bychom museli vyhodnotit charakteristickou funkci $1_{f(x)=f(z)}$ pro všechna $z \in \mathcal{D}_x(\cdot|A)$, což není se spojitými hodnotami nebo v případě velkých vstupních prostorů možné, zavádí autoři pravděpodobnostní definici precision:

$$P(prec(A) \geq \tau) \geq 1 - \delta$$

Díky tomu můžeme hledání vyhovujícího pravidla definovat jako optimalizační kombinatorický problém:

$$\max_{A \text{ s.t. } P(prec(A) \geq \tau) \geq 1-\delta} cov(A)$$

Tedy hledání pravidla, které má nejvyšší možnou coverage a zároveň splňuje, že $prec(A) \geq \tau$ s pravděpodobností alespoň $1 - \delta$. Problém je, že počet kandidátních pravidel roste exponenciálně, takže nelze řešit tuto úlohu exaktně.



Obrázek 3.14: Algoritmus Anchors[12].

Základem algoritmu (znázorněn na obrázku 3.14) je tzv. beam search (paprskové prohledávání), což je grafový algoritmus, kombinující prohledávání do šířky (depth-first) a do hloubky (breadth-first). Zde funguje tak, že v první

„generaci“ (opakování cyklu algoritmu) se vytvoří n kandidátních pravidel – pro každou vstupní proměnnou jedno, s jedním predikátem. V každém dalším kole se přenáší a rozšiřuje o další predikát pouze B nejlepších kandidátů, tj. kandidátů s nejvyšší precision. V i-tém kole tedy algoritmus pracuje s pravidly s právě i predikáty, ze kterých na konci vybere nejlepších B, které použije v dalším kole. Zároveň se tím určuje i horní mez počtu cyklů, n – počet vstupních proměnných. V n-tém kole totiž algoritmus pracuje s pravidlem s n predikáty a precision takového pravidla bude 1, protože přesně popisuje právě vysvětlovanou instanci.

Dalším ústředním bodem algoritmu je vyhodnocování precision a nalezení nejlepších B pravidel. Pro její vyhodnocení musíme prozkoumat okolí D zkoumané instance, což znamená volat vysvětlovaný model pro různé instance. Jelikož tato činnost bývá výpočetně náročná a jedná se vlastně o pure exploration multi-armed bandit problém, který je jedním z klasických reinforcement learning problémů, používají autoři algoritmus KL-LUCB[25].

Označení multi-armed bandit je analogie k případu, kdy hráč stojí před řadou klasických, pákových, výherních automatů, přičemž každý má jiné šance na výhru. Cílem je získat co nejvyšší výhru s omezenými zdroji. Každé zatažení páky stojí nějaké zdroje, ale poskytne lepší představu o výhodnosti automatu. Algoritmus tedy volí kompromis mezi prozkoumáváním a vytěžováním nejlepšího dosud nalezeného automatu. V případě Anchors je každé kandidátní pravidlo A pákou automatu (arm) a její „zatažení“ vyhodnotí sousedy pro dané pravidlo, čímž získá lepší představu (s vyšší konfidenční hladinou) o precision pravidla A.

3.7.3 Výhody Anchors

- Výstupem Anchors jsou IF-ELSE pravidla, která jsou velmi snadno interpretovatelná i laiky.
- Stejná pravidla lze používat i pro více instancí. Pro jak velkou část instancí bude pravidlo aplikovatelné ukazuje míra coverage.
- Anchors, respektive multi-armed bandit část, je dobře paralelizovatelná.

3.7.4 Nevýhody Anchors

- Součástí Anchors jsou nejen parametry, které je potřeba odladit tak, aby výsledek odpovídal našemu očekávání (parametr B pro beam search, precision threshold τ), ale také je potřeba zamyslet se, co v naší doméně znamená okolí instance. Podobně jako u LIME je problém tyto sousedy vytvořit tak, abychom respektovali možné vlastnosti reálných dat (různé korelace apod.).

- Vstupní, ale v případě regrese i výstupní, data je většinou nutné diskretizovat. Pokud to neuděláme budou výsledná pravidla mít příliš malou coverage, tj. budou příliš konkrétní.

- Anchors vyžaduje mnoho volání modelu, pro složité modely může tedy být tato metoda nevhodná.

3.8 Shapleyho hodnoty

Shapleyho hodnota je hodnota, pomocí které lze v kooperativní hře přiřadit zisk jednotlivým hráčům na základě jejich přínosu. Pojmenována je podle Lloyda S. Shapleyho, který ji představil v roce 1953[26].

Pokud Shapleyho hodnoty používáme v otázce interpretability pak hrou myslíme predikci pro jednotlivou instanci, ziskem hodnotu predikce (výstup modelu f) a hráči jsou jednotlivé proměnné. Ty při *hře* spolupracují a ve výsledku dosáhnou *zisku* o hodnotě predikce. A Shapleyho hodnota proměnné (hráče) je její průměrný marginální přínos přes všechny možné koalice.

Můžeme si představit, že máme model, který predikuje hodnotu bytu, pro jednoduchost, na základě 4 vlastností – obytná plocha, patro, zda je v blízkosti park a zda jsou v bytě povoleny kočky. V konkrétním případě bytu s obytnou plochou 50m^2, ve druhém podlaží, v blízkosti parku a bez možnosti koček predikuje model hodnotu bytu na 3,1 milionu korun. Shapleyho hodnoty vysvětlují rozdíl mezi průměrnou predikcí, řekněme 3 miliony Kč, a touto predikcí, tedy 100000 Kč. Pro tento příklad mohou Shapleyho hodnoty vypadat třeba následovně:

- obytná plocha = 50m^2 – +200000
- patro = 2 – 0
- park blízko = true – +50000
- kočky povoleny = false – −150000.

3.8.1 Příklad

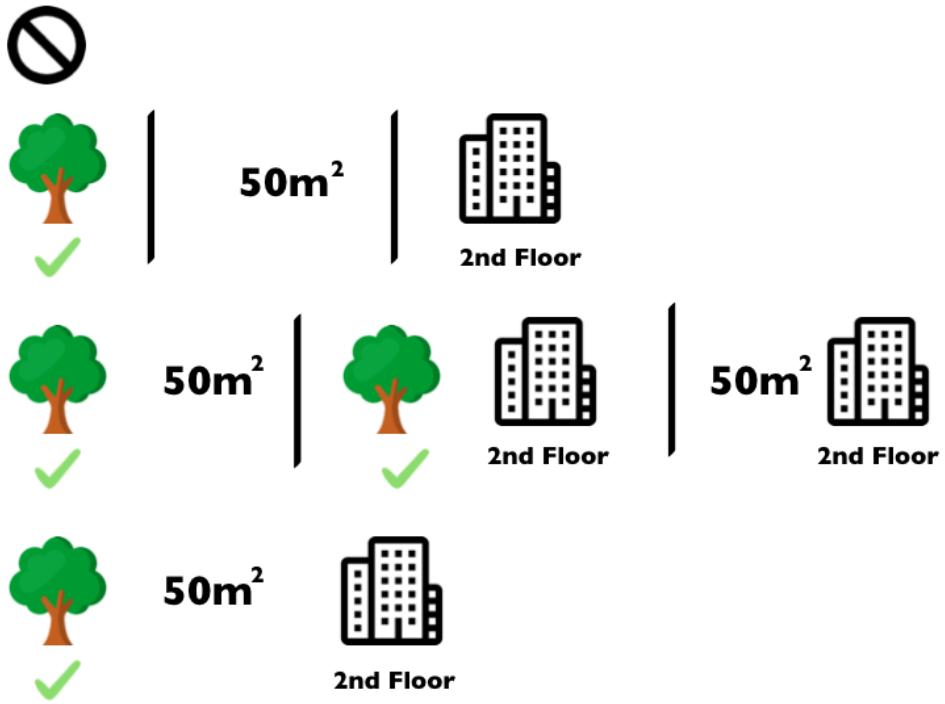
Výše jsem zmínil, že Shapleyho hodnota je průměrný marginální přínos proměnné (hráče) přes všechny možné koalice. Příkladem se pokusím ilustrovat, co to znamená.

Koalicí se v teorii her rozumí skupina hráčů spolupracujících při volbě strategií, případně při přerozdělování výhry. Shapleyho hodnota zkoumá přínos určité proměnné do koalice tak, že srovnává zisk (predikci) v případě, že proměnná je v koalici zahrnuta a v případě, že není.

Například v případě, kdy zkoumáme přínos zákazu koček do koalice s blízkostí parku a obytnou plochou 50m^2 nejprve zjistíme predikci, pro případ, že zákaz koček v koalici je. Hodnoty zbylých proměnných nastavíme náhodně na základě výběru z dat. Pro tento konkrétní případ můžeme tedy získat instanci:

- obytná plocha = 50m^2
- patro = 2
- park blízko = false
- kočky povoleny = false.

Poté odebereme zkoumanou proměnnou z koalice, tj. nahradíme ji náhodně vybranou hodnotou z dat (může být i stejná). Rozdíl v predikcích s a bez proměnné v koalici je pak přínos této konkrétní koalici. Tyto kroky můžeme opakovat a průměrovat, čímž dostaneme lepší odhad pro tuto koalici, nicméně abychom získali Shapleyho hodnotu, musíme provést toto zkoumání pro *všechny možné koalice* a výsledky zprůměrovat. Všechny koalice, pro které musíme provést analýzu jsou ilustrovány na obrázku 3.15



Obrázek 3.15: Všechny koalice pro proměnnou „kočky povoleny“ [12].

3.8.2 Interpretace Shapleyho hodnot

Interpretace Shapleyho hodnoty pro proměnnou j je hodnota, kterou konkrétní ohodnocení proměnné j přispělo (jak ovlivnilo) k predikci této instance oproti průměrné predikci pro celý dataset.

Pohled na Shapleyho hodnotu může svádět ke špatné interpretaci ve smyslu, *jak by odebrání proměnné j ovlivnilo výslednou predikci*. To ale není správná interpretace. Takovou informaci Shapleyho hodnota vůbec neposkytuje.

3.8.3 Odhad Shapleyho hodnot

Vzhledem k tomu, že počet možných koalic roste exponenciálně s počtem proměnných, není vždy únosné, počítat Shapleyho hodnotu exaktně. Jednu z možností, jak tuto hodnotu aproximovat pomocí metody Monte Carlo navrhli Strumbelj a Konomenko[27]:

$$\hat{\phi}_j = \frac{1}{M} \sum_{m=1}^{M} \left(\hat{f}(x_{+j}^m) - \hat{f}(x_{-j}^m) \right)$$

kde $\hat{f}(x)$ je predikce pro x, vektor x_{+j}^m je vektor x s náhodným počtem proměnných jejichž hodnoty byly nahrazeny hodnotami z jiné instance z (kromě proměnné j, která určitě zůstala nenahrazena). Vektor x_{-j}^m je téměř identický vektor, pouze s tou změnou, že i hodnota proměnné j byla nahrazena hodnotou z instance z.

3.8.4 Výhody Shapleyho hodnot

- Rozdíl mezi průměrnou predikcí a vysvětlovanou predikcí je vždy plně rozdělen mezi proměnné. To zaručuje tzv. efficiency vlastnost Shapleyho hodnoty ($\sum_{j=1}^{p} \phi_j = \hat{f}(x) - E_X(\hat{f}(X))$). Vysvětlení je tedy úplné a rozložené přes všechny proměnné.
- Shapleyho hodnoty umožňují vytvoření kontrastního vysvětlení. Místo porovnávání s průměrnou predikcí můžeme srovnávat vliv vůči jiné instanci nebo nějaké množině instancí.

3.8.5 Nevýhody Shapleyho hodnot

- Shapleyho hodnoty jsou náročné na výpočet, proto nám většinou musí stačit aproximace.
- Shapleyho hodnota může svádět ke špatné interpretaci, viz 3.8.2.
- Vysvětlení pomocí Shapleyho hodnot bude vždy používat všechny proměnné, což může v některých případech vést k nižší přehlednosti. Navíc pro člověka z pravidla bývá preferovanější vysvětlení za pomoci jen několika málo důležitých proměnných.
- Podobně jako u většiny ostatních metody se v případě, kdy jsou proměnné korelované, mohou používat pro vysvětlení i instance, které nejsou možné, nebo jsou velmi nepravděpodobné.

3.9 SHAP

SHAP (Shapley additive values) je lokální metoda pro vysvětlení jedné instance, založená na Shapleyho hodnotách. Tato metoda částečně kombinuje

LIME a Shapley values, jelikož Shapleyho hodnoty, které instanci vysvětlují tvoří lineární model.

3.9.1 Definice

SHAP definuje vysvětlení jako

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$

g je vysvětlující model, $z' \in \{0,1\}^M$ je „simplified features“ vektor, tj. vektor určující, zda je proměnná v koalici přítomna, nebo ne (1 znamená, že příslušná proměnná v koalici je, 0, že není). M označuje maximální velikost koalice a $\phi_j \in \mathbb{R}$ je Shapleyho hodnota pro proměnnou j.

Pro samotnou instanci x se simplified features vektor zjednoduší na vektor jedniček a vysvětlení tedy bude

$$g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j$$

3.9.2 Vlastnosti SHAP

Lundberg a Lee[28] definují ve své práci představující SHAP, 3 hlavní vlastnosti, které SHAP splňuje

1. Local accuracy

   $$f(x) = g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j$$

   Tato vlastnost zaručuje, že vysvětlující model $g(x')$ bude mít totožný výstup s vysvětlovaným modelem $f(x)$, pro „zjednodušený“ (simplified) vstup x'. Zároveň, pokud položíme $\phi_0 = E_X(\hat{f}(x))$ a nastavíme všechny x'_j na 1 dostaneme efficiency vlastnost Shapleyho hodnot:

   $$f(x) = \phi_0 + \sum_{j=1}^{M} \phi_j x'_j = E_X(\hat{f}(X)) + \sum_{j=1}^{M} \phi_j$$

2. Missingness

   $$x'_j = 0 \Rightarrow \phi_j = 0$$

   Missingness říká, že proměnná, která není v originálních datech nebude mít žádný vliv. Tato vlastnost je potřebná, protože teoreticky by taková proměnná mohla mít jakoukoli Shapleyho hodnotu, aniž by ovlivnila local accuracy (protože by vždy byla násobená $x'_j = 0$).

3. Consistency

   Nechť $f_x(z') = f(h_x(z'))$ a $z_{\setminus j'}$ značí, že $z'_j = 0$, pro jakékoli 2 modely f a f', pro které platí

   $$f'_x(z') - f'_x(z'_{\setminus j}) \geq f_x(z') - f_x(z'_{\setminus j})$$

   pro všechna $z' \in \{0,1\}^M$, pak:

   $$\phi_j(f', x) \geq \phi_j(f, x)$$

   Tato vlastnost říká, že pokud se přínos proměnné j v modelu zvýší nebo zůstane stejná bez ohledu na ostatní proměnné, pak její ϕ_j zůstane také alespoň stejná.

3.9.3 Kernel SHAP

Kernel SHAP je model agnostický algoritmus pro SHAP, který odhaduje přínos každé proměnné. Skládá se z pěti kroků:

1. Vytvoř K náhodných simplified features vektorů $z'_k \in \{0,1\}^M$, $k \in \{1, \ldots, K\}$.

2. Získej predikci modelu pro každý z'_k pomocí funkce $h_x(z'_k)$, která převede vektor z'_k do původního prostoru proměnných. $h_x()$ nahradí každou 1 původní hodnotou proměnné z vysvětlované instance x a 0 hodnotou proměnné z náhodně vybrané instance.

3. Každému z'_k přiřaď váhu pomocí tzv. SHAP kernel:

   $$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)}$$

   M je maximální velikost koalice (délka simplified features vektoru) a $|z'|$ je počet jedniček ve vektoru z'.

   Na rozdíl od přístupu LIME přiřazuje kernel SHAP vyšší váhu nejen vektorům s vyšším počtem 1, ale právě i naopak s co nejnižším. Ve své podstatě říká, že nejvíce se naučíme o jednotlivých proměnných právě tím, že je izolujeme – nahradíme téměř všechny ostatní nebo naopak jen ji.

4. Natrénuj vážený lineární model $g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$. Optimalizujeme tedy ztrátovou funkci L: $L(f, g, \pi_x) = \sum_{z' \in Z}[f(h_x(z')) - g(z')]^2 \pi_x(z')$.

5. Vrať Shapleyho hodnoty ϕ_k – koeficienty lineárního modelu g.

3.9.4 Další SHAP metody

Autoři také navrhli i další SHAP metody, pro specifické modely, jako Deep SHAP nebo Tree SHAP[29]. Tyto metody poskytují v případech, kdy je lze použít zásadní vylepšení a proto je lepší využívat právě je.

3.9.5 Výhody SHAP

- SHAP si ponechává všechny výhody a snad kromě výpočetní náročnosti u specifických metod jako Tree SHAP i nevýhody Shapleyho hodnot.
- SHAP unifikuje oblast interpretability modelů strojového učení, jelikož kombinuje LIME s Shapleyho hodnotami.

3.9.6 Nevýhody SHAP

- Největší nevýhodou Kernel SHAP je výpočetní náročnost, která brání většímu využití a také využívání SHAP pro globální interpretabilitu, která by vyžadovala vypočítání Shapleyho hodnot pro mnoho instancí.
- Stejně jako u ostatních permutačních metod dochází k používání nevhodných instancí, pokud jsou proměnné korelované.

3.10 Counterfactual explanation (vysvětlení protipříkladem)

Jednou z dalších možností, jak vysvětlit, nebo alespoň přiblížit chování modelu je pro konkrétní instanci ukázat, co by se muselo změnit, aby model změnil svou predikci. Takové vysvětlení nám může poskytnout i dodatečnou informaci o tom, co bychom měli změnit, aby predikce pro nás byla příhodná. Například pokud nám v bance odmítnou úvěr na základě výstupu modelu, pak vysvětlení protipříkladem říká, co bychom měli udělat, abychom úvěr dostali.

Cílem je tedy získat co nejpodobnější instanci, kterou model ohodnotí jiným výstupem, neboli najít co nejmenší změnu, která ovlivní predikci modelu natolik, že se změní. Toho lze dosáhnout jednoduchým prohledáváním okolí instance.

Hlavní požadavky na protipříklad jsou 3:

- Model ohodnotí protipříklad požadovanou predikcí – v případě klasifikace můžeme požadovat určitou třídu, případně jakoukoli jinou než původní. U regrese je možné stanovit nějaký limit, kterého musí protipříklad dosáhnout.
- Protipříklad je co nejpodobnější vysvětlované instanci – pro tento požadavek potřebujeme stanovit nějakou míru podobnosti. Jakou míru použít

záleží na konkrétním případě, ale většinou bývá žádoucí, aby se změnilo co nejméně proměnných.

- Protipříklad by měl být pravděpodobnou instancí – pokud by protipříkladem byla instance, která nedává smysl, např. byt se zápornou obytnou plochou, pak nám neposkytuje užitečnou informaci.

3.10.1 Generování protipříkladů

Nejjednodušší možností je naivní prohledávání okolí náhodným měněním hodnot proměnných. To je sice naprosto validní přístup, ale samozřejmě existují lepší možnosti.

Jednou z nich je minimalizovat tuto ztrátovou funkci:

$$L(x_i, x', y', \lambda) = \lambda \cdot (\hat{f}(x') - y')^2 + d(x_i, x')$$

první část je čtverec rozdílu mezi predikcí modelu $\hat{f}$ pro protipříklad x' a požadovaným výstupem y'. Druhá část je vzdálenost d vysvětlované instance x_i a protipříkladu x'. Parametr λ určuje rovnováhu mezi dosažením požadovaného výstupu modelu a tím, že protipříklad je podobný původní instanci[30].

Místo pevně zvoleného parametru λ, ale autoři navrhují spíše zvolit toleranci ϵ – $|\hat{f}(x') - y'| \leq \epsilon$ a formulaci optimalizačního problému

$$\arg\min_{x'} \max_{\lambda} L(x, x', y', \lambda)$$

který můžeme řešit jakýmkoli dostupným optimalizátorem.

Pro vzdálenostní míru d zvolili autoři Manhattanskou vzdálenost váženou střední hodnotou směrodatné odchylky (median absolute deviation – MAD):

$$d(x, x') = \sum_{j=1}^{p} \frac{|x_j - x'_j|}{MAD_j}$$

Tento přístup dále rozvíjí Dhurandhar et al.[31] přidáním autoenkóderu a tzv. elastic net regularizeru a Looveren a Klaise[32] využitím prototypů. Výsledná ztrátová funkce pak vypadá následovně:

$$L = \lambda \cdot L_{pred} + \beta \cdot L_1 + L_2 + L_{AE} + L_{proto}$$

kde

$$L_{pred} = \max([Pred(x_0 + \delta)]_{t_0} - \max_{i \neq t_0}[Pred(x_0 + \delta)]_i, -\kappa)$$

$[Pred(x_0 + \delta)]$ je predikce třídy i pro protipříklad $x_0 + \delta$ a κ je konfidenční parametr.

$$\beta \cdot L_1 + L_2 + L_{AE} = \beta \cdot \|\delta\|_1 + \|\delta\|_2^2 + \gamma \cdot |x_0 + \delta - AE(x_0 + \delta)\|_2^2$$

$\beta \cdot L_1 + L_2$ se souhrnně nazývá elastic net regularizátor a L_{AE} symbolizuje L_2 reconstruction error pro x_{cf}.

Definici L_{proto} a další optimalizace lze nalézt v [32].

3.10.2 Výhody protipříkladů

- Vysvětlení protipříkladem je jednoduché. Neříká, ale ani se nesnaží říkat, nic jiného než jak by vypadala podobná instance, která by ale měla jinou predikci. Na rozdíl od např. LIME, což je lokální metoda, která ale může svádět k misinterpretaci přenášením lokálních vztahů na celý model.
- Protipříklady příhodně odpovídají lidskému přemýšlení a poskytují zajímavou informaci. Většinou je totiž pravděpodobné, že budeme schopni najít takový protipříklad, který mění jen vlastnosti, které jsme schopni ovlivnit.

3.10.3 Nevýhody protipříkladů

- Pro každou instanci bude existovat několik protipříkladů a není snadné vybrat právě ten „nejlepší“ pro danou příležitost. Navíc i pro stejnou instanci se v různých situacích mohou hodit různé protipříklady.
- Neexistuje garance, že pro zadanou míru tolerance ϵ dokážeme vůbec nějaký protipříklad nalézt.

Kapitola 4

Softwarové nástroje interpretability modelů strojového učení

Pro interpretabilitu modelů samozřejmě existuje řada knihoven implementující nejen zde uvedené metody. Nejčastěji se jedná o knihovny v jazycích R a Python, jelikož se jedná o 2 nejpoužívanější programovací jazyky v prostředí data science. Dobrým zdrojem nejen pro dohledání softwarových nástrojů, ale i jiných zdrojů a textů týkajících se problematiky interpretability modelů strojového učení je *Awesome compendium*[33].

4.1 iml: interpretable machine learning

iml[34] je knihovna v jazyce R, zaměřující se na agnostické metody interpretability. Implementuje velkou část zde uvedených metod, od feature importance a vizualizačních metod jako PD grafy až po LIME nebo SHAP. Nevýhodou této knihovny je nicméně chybějící dokumentace. Jediným zdrojem, který se mi podařilo dohledat je ukázkový Jupyter notebook, ze kterého musí uživatel trochu odhadovat, jak přesně se která metoda používá. Případným dalším zdrojem může být samotný zdrojový kód, který ale není sám o sobě zrovna *self-documenting*.

4.2 DALEX

Další souhrnnou knihovnou v jazyce R je DALEX[35]. Tato knihovna nabízí zajímavé možnosti vizualizací, jako např. *Ceteris Paribus Profiles*, v podstatě PD graf pro jednu danou instanci, *Break Down attributions* a *LIVE – Local Interpretable Visual Explanations*, obojí metody založené na LIME, popsané v [36], SHAP a další metody.

Jednou z hlavní výhod této knihovny je to, že je díky předpřipraveným wrapperům, buď přímo v knihovně DALEX nebo v rozšíření DALEXtra, schopná pracovat s modely nejen z různých knihoven, ale i jazyků (např. sci-kit modely v Pythonu nebo h2o v Javě). Nicméně uživatel nemusí tyto wrappery využít a může si vytvořit vlastní. DALEX tedy prakticky funguje jako adaptér mezi jednotlivými knihovnami implementujícími samotné metody a uživatelskými modely.

4.3 Skater

Skater[37] je framework knihovna v jazyce Python, vznikající pod záštitou společnosti Oracle, která, byť je zatím stále v *beta* verzi a její vývoj byl na delší dobu přerušen, nabízí, nebo má alespoň potenciál do budoucna nabízet, unifikovaný přístup k interpretaci black-box modelů. Zatím implementuje feature importance, PD grafy, LIME, metody pro interpretaci komplexních neuronových sítí jako Layer-wise Relevance Propagation a Integrated Gradient nebo surrogate modely.

Nevýhodou je ovšem neudržovanost knihovny, vedoucí k problémům při instalaci a celkovou nejistotu, zda a jak dobře bude knihovna nadále použitelná v budoucnosti. Z těchto důvodů bych momentálně, dokud nedojde k významnějšímu zásahu, ať už ze strany autorů nebo samotného Oracle, nedoporučoval tuto knihovnu využívat.

4.4 Alibi

Další Python knihovnou pro interpretabilitu je knihovna Alibi[38], nabízející implementaci Anchors, Contrastive Explanation Method a Counterfactuals (včetně verze s prototypy). Díky kvalitní dokumentaci s několika ukázkami použití je práce s touto knihovnou celkem snadná.

Bohužel, v některých případech může být problém, že tato knihovna, kvůli závislosti na knihovně blis, lze používat pouze s 64bitovou verzí Pythonu.

4.5 AI Explainability 360

AI Explainability 360[39] je open-source Python knihovna zaštítěná společností IBM, nabízející řadu nejnovějších algoritmů z oblasti interpretability. Nabízí celkem 8 black-box, ale i white-box algoritmů, mezi které patří LIME, SHAP nebo TED explainer – framework, kdy se k trénovacímu datasetu připojí i vysvětlení a model se učí předpovídat nejen správný výstup (Y), ale i vysvětlení pro něj (E), viz [40]. Knihovna má velmi dobrou dokumentaci, ale také pěkné představení včetně videí, odkazů na zdrojové akademické práce a dokonce i demo zobrazitelné přímo v prohlížeči[41].

Nevýhodou ovšem může být to, že oficiálně je knihovna podporovaná pouze pro Python verze 3.6[39]. To lze ale řešit virtuálním prostředím, pro které má Python přímo zabudovanou podporu a je to i doporučený postup instalace.

Pro algoritmy férové AI existuje také sesterská knihovna AI Fairness 360[42].

4.6 Ostatní knihovny

Samozřejmostí je pak řada knihoven implementujících jednotlivé algoritmy. Mnohdy se jedná přímo o knihovny od autorů prací představujících daný algoritmus. Příkladem takové knihovny může být knihovna lime[43], implementující LIME jak pro tabulková data, tak textová, či obrázková. Dokonce nechybí ani podpora pro rekurentní neuronové sítě.

Podobným příkladem knihovny přímo od autorů algoritmu je knihovna shap[44], která nabízí nejen SHAP a treeSHAP, ale i přehledné, v některých případech i interaktivní, vizualizace výsledků ve formě tzv. *force* grafů nebo *decision* grafů.

Další knihovnou, která stojí za zmínku je PDPbox[45] pro PD grafy a další užitečné vizualizace, včetně ICE grafů.

Celkově si myslím, že pro komplexní analýzu interpretability modelu je vhodné jednotlivé knihovny a frameworky kombinovat. Jednoduše využít to dobré z mnoha dostupných možností.

Kapitola 5

Použití metod interpretability

V této části práce ukážu, jak by mohla vypadat analýza interpretovatelnosti jednoduchých ukázkových modelů.

5.1 Titanic

5.1.1 Data

Pro první příklad jsem zvolil dobře známý, alespoň ve světě machine learningu, dataset Titanic. Na něm tedy ukáži jak by mohla vypadat analýza jednoduchého klasifikátoru.

Jedná se o data o cestujících z lodi Titanic, která poskytuje společnost Kaggle[46]. Cílem je na základě poskytnutých znalostí o cestujícím předpovědět, zda první a zároveň poslední cestu této „nepotopitelné“ lodi přežil. K tomu máme k dispozici následující proměnné:

proměnná	**význam**
Survived	Zda cestující přežil – cílová proměnná
PassengerId	Unikátní identifikátor cestujícího
Pclass	Třída, ve které cestoval
Name	Jméno
Sex	Pohlaví
Age	Věk
SibSp	Počet manželek/ů, či sourozenců, se kterými cestoval
ParCh	Počet rodičů/dětí, se kterými cestoval
Ticket	Číslo lístku
Fare	Cena, kterou cestující zaplatil za lístek
Cabin	Číslo kajuty
Embarked	Kde se cestující nalodil

5.1.2 Model

Po provedení drobného předzpracování dat vstupují do modelu následující proměnné:

Proměnná	**Význam**
Pclass	Třída, ve které cestoval
Sex	Pohlaví
Age	Věk
SibSp	Počet manželek/ů, či sourozenců cestujících společně
ParCh	Počet rodičů/dětí cestujících společně
Fare	Cena, kterou cestující zaplatil za lístek
Embarked	Kde se cestující nalodil
Title	Titul cestujícího (např. Mr)

Nad těmito daty jsem natrénoval neuronovou síť z knihovny sklearn s následujícími parametry:

```
MLPClassifier(solver='lbfgs', alpha=1e-5,
↪  hidden_layer_sizes=(5, 2), random_state=1)
```

Tento model dosahuje přesnosti (accuracy) přibližně 0, 78, což zdaleka není nijak závratné číslo, nicméně cílem této práce není vytvořit nejlepší možný model, takže nám nevadí, že nedosahuje produkčních kvalit. Naopak si můžeme jeho interpretací zkusit ukázat, kde dělá největší chyby.

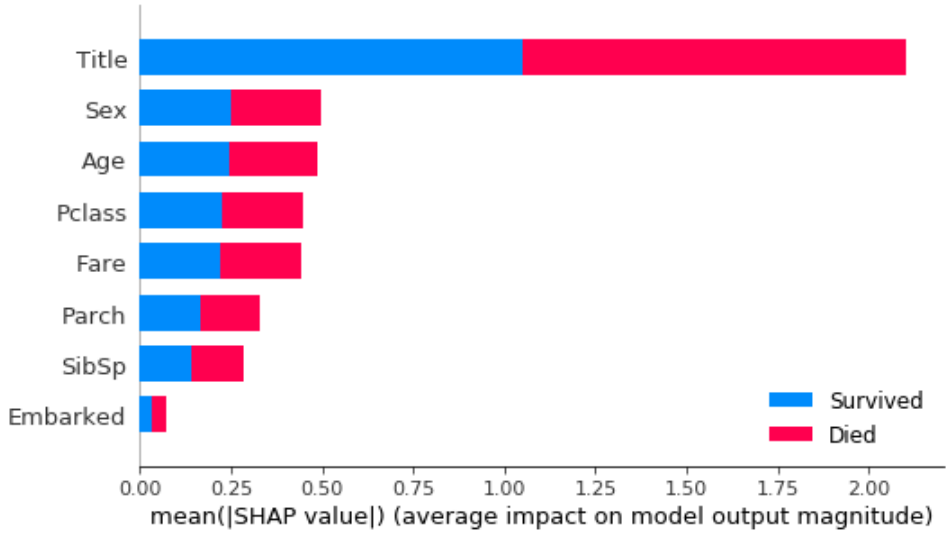
5.1.3 Feature importance

Pomocí permutation feature importance, získanou za použití knihovny *eli5* [47], dostaneme první náhled na to, které proměnné jsou pro model důležité. Pro tuto metodu jsem použil trénovací data, jelikož poskytovaná testovací množina nemá cílovou proměnnou (protože se jedná o „soutěžní“ úlohu) a model jsem trénoval nad všemi trénovacími daty,kterých je všehovšudy pouze 891 záznamů. V takovém případě bychom samozřejmě mohli dogenerovat další data apod. nicméně to opět není předmětem této práce.

Výstup metody může vypadat přibližně následovně (jednotlivé běhy se mohou lišit, kvůli nedeterminističnosti metody):

Váha	**Proměnná**
0.1881 ± 0.0163	Title
0.0224 ± 0.0103	Sex
0.0213 ± 0.0110	Fare
0.0155 ± 0.0114	Age
0.0130 ± 0.0054	Parch
0.0040 ± 0.0070	SibSp
0.0018 ± 0.0044	Embarked
-0.0007 ± 0.0042	Pclass

Další možností, jak získat feature importance je použití metody SHAP, tedy získání Shapleyho hodnot a seřazení proměnných podle její průměrné absolutní hodnoty, jinými slovy průměrného vlivu na výstup modelu. Tuto úlohu řeší knihovna shap[44], přičemž její výsledek se, přes drobné odlišnosti, od permutation feature importance neliší nijak výrazně:



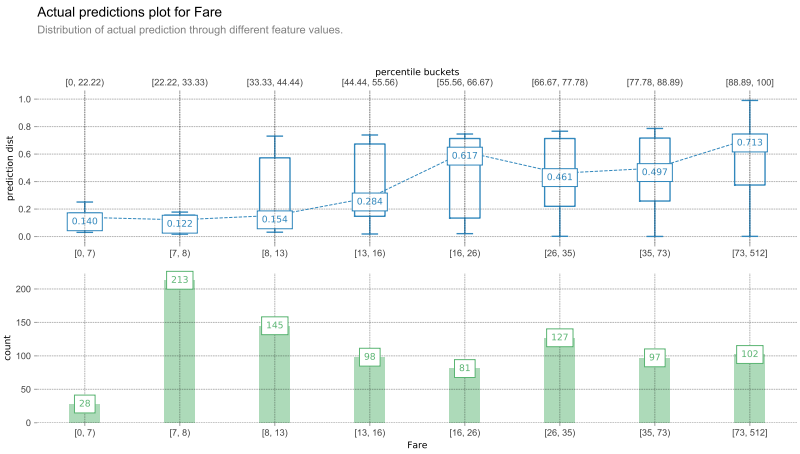
Obrázek 5.1: Feature importance pomocí metody SHAP.

Již na základě této jednoduché analýzy můžeme prohlásit, že se model až příliš zaměřuje pouze na proměnnou Title. To sice dává smysl i z hlediska zdravého rozumu, jelikož v této kategorické proměnné se skrývá pohlaví a sociální postavení, tedy i majetnost cestujícího apod. Nicméně pro dosažení větší přesnosti by bylo určitě vhodné čerpat více znalostí i z ostatních proměnných, přece jen rozdíl mezi Title a druhou nejdůležitější proměnnou Sex (které jsou mimochodem velmi korelované) je propastný.

Zajímavostí je i úsměvná negativní váha u proměnné Pclass, značící, že když u instancí změníme náhodně zaměníme hodnoty této proměnné, tak se přesnost modelu může zvýšit. Nicméně toto číslo je tak malé, že se jedná o nějakou odchylku a vlastně tato váha spíše říká, že tuto proměnnou model téměř nebere v potaz.

5.1.4 Partial dependence grafy

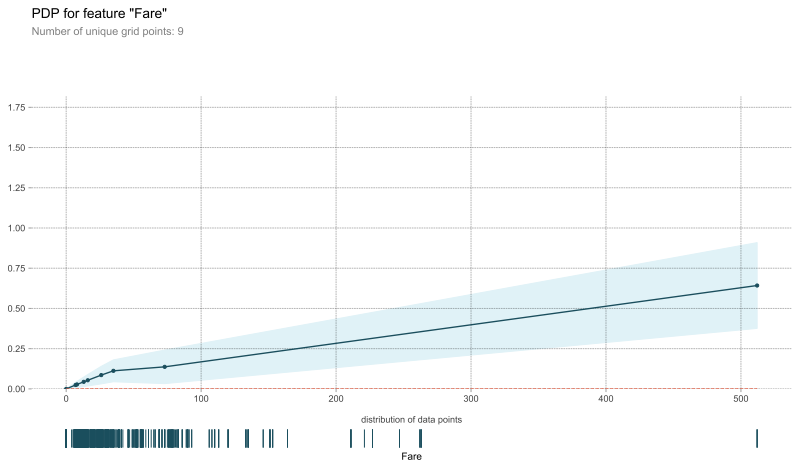
Pro PD grafy jsem využil knihovnu PDPbox[45], pomocí které lze vykreslit i ICE grafy a další vizualizace, jako například *actual plot* – graf, zobrazující jak vypadají predikce pro instance s určitými hodnotami proměnné, viz obrázek 5.2.



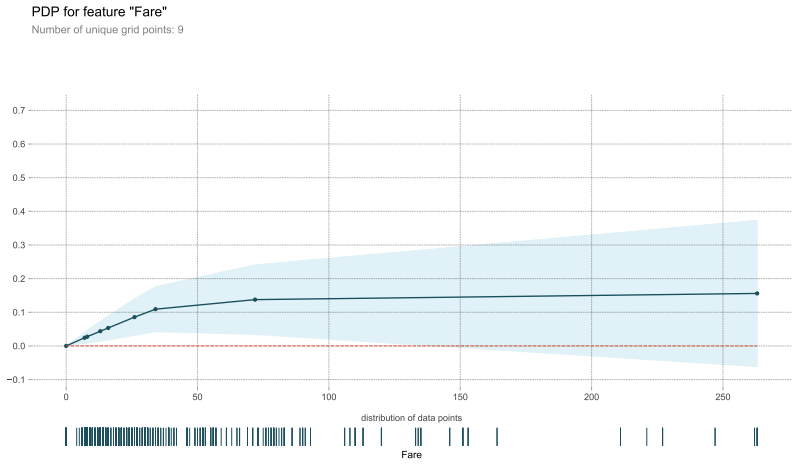
Obrázek 5.2: *Actual plot* predikcí proměnné Fare.

Obrázek 5.3 ukazuje partial dependence graf pro proměnnou Fare. Z tohoto grafu můžeme vidět, že vztah Fare (tj. ceny za lístek) a predikce modelu je přibližně takový, jaký bychom mohli očekávat. S rostoucí Fare roste i predikce šance na přežití. U tohoto grafu si ale musíme dávat pozor na distribuci hodnot. Mezi hodnotou cca 250 a hodnotou přes 500 ve skutečnosti neexistují žádné instance, takže informace, kterou graf podává nemusí být přesná. To ukazuje graf na obrázku 5.4, který zobrazuje pouze instance s Fare menší než 300. Díky tomuto srovnání vidíme, že vlastně od určité hodnoty už vyšší Fare nezvyšuje predikovanou šanci na přežití a že graf 5.3 není zcela přesný a vypovídající.

Drobnou nevýhodou knihovny PDPbox je, že neumí přepnout mezi spojnicovým (čárovým) a sloupcovým grafem, to může být nevýhoda pro kate-

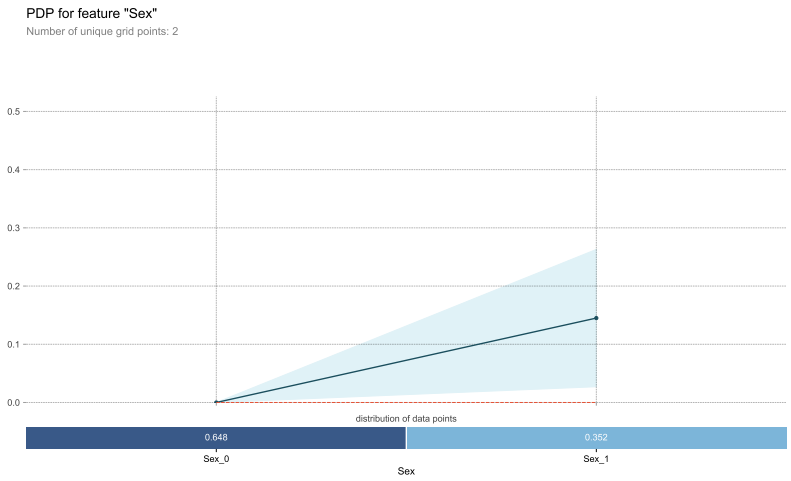


Obrázek 5.3: Partial dependence graf pro proměnnou Fare.



Obrázek 5.4: Partial dependence graf pro proměnnou Fare s omezením Fare < 300.

gorické proměnné, u kterých se budeme muset spokojit s grafem, kde bude spojnice mezi jednotlivými hodnotami kategorické proměnné, přestože taková spojnice nedává smysl, viz obrázek 5.5. Navíc pokud není proměnná one-hot encoded s vypovídajícím názvem, pak bude nedokonalý i popisek grafu, opět ilustrováno grafem 5.5.



Obrázek 5.5: Partial dependence graf pro proměnnou Sex. „Sex_0“ symbolizuje muže a „Sex_1“ ženy.

5.1.5 LIME

5.1.5.1 Instance

LIME, ale i další metody, které budou následovat, patří do skupiny lokálních metod, tj. vysvětlují jednu predikci nebo skupinu predikcí. Proto je potřeba nějakou takovou instanci si vybrat. V praxi může takový požadavek přijít např. od zákazníka, kterému byla zamítnuta půjčka. Nicméně při vývoji a, jako v našem případě, experimentálních testech si musíme zvolit instance sami, případně je i vytvořit. Je vhodné volit jak výrazně odlišné instance, pro které je třeba i predikce různá, tak ale i některé podobné, pro kontrolu stability modelu, či zvolených metod a tedy ověření, že získanému závěru můžeme věřit.

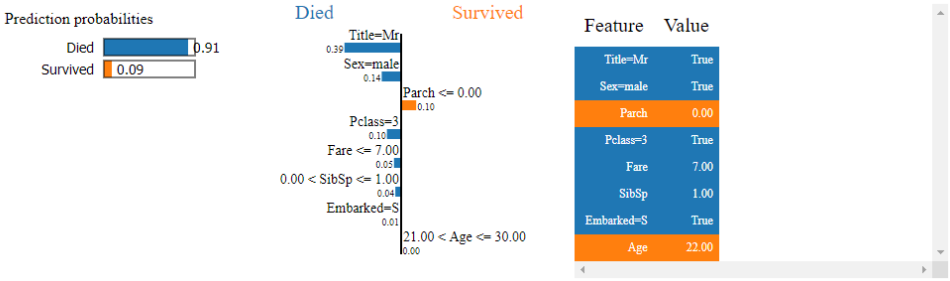
Pro ukázku v případě této práce jsem nicméně zvolil jen 2 instance, popsané v tabulce 5.1. Další možnosti lze prozkoumat pomocí interaktivního Jupyter notebooku *Titanic.ipynb* v přílohách práce.

proměnná	**Instance 1**	**Instance 2**
Pclass	3	1
Sex	0 (muž)	1 (žena)
Age	22	38
SibSp	1	1
ParCh	0	0
Fare	7	71
Embarked	1 (Southampton)	0 (Cherbourg)
Title	0 (Mr)	1 (Mrs)

Tabulka 5.1: Zkoumané instance datasetu Titanic.

5.1.5.2 Výsledky metody

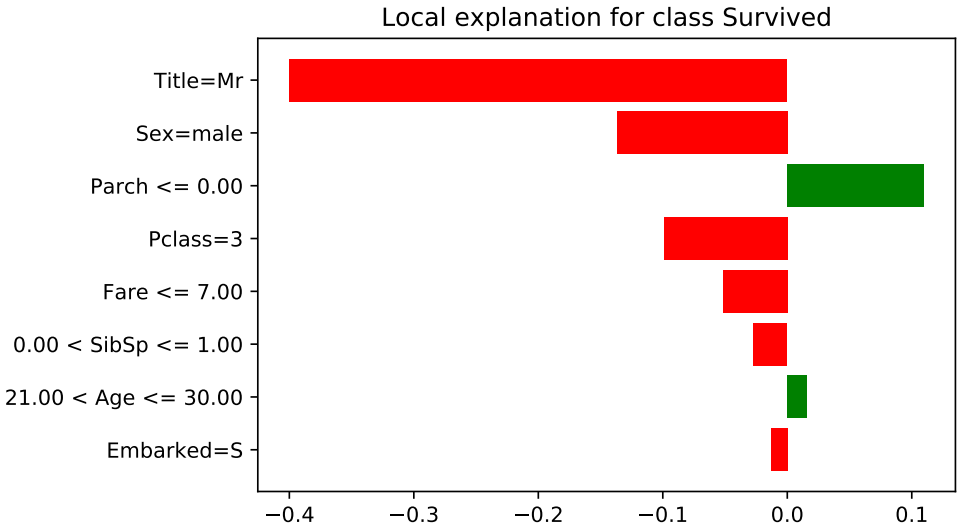
Pro metodu LIME jsem využil výše zmíněnou knihovnu lime[43] od autorů článku o metodě samotné. Tato knihovna poskytuje vysvětlení ve formě html, zobrazitelné i přímo v Jupyter notebooku, viz obrázek 5.6 nebo jako *pyplot figure*, tedy graf ve formátu knihovny *pyplot*, viz obrázek 5.7. Všechna vysvětlení byla generována se základním nastavením kernel width, tedy $0,75 \times \sqrt{počet\ sloupců}$, tj. $0,75 \times \sqrt{8} \approx 2,12$.



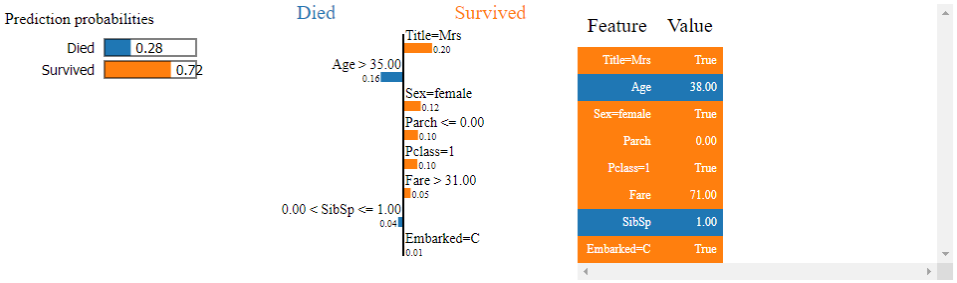
Obrázek 5.6: LIME vysvětlení pro 1. instanci v html formátu.

Co se obsahu obrázků týče, ukazují vysvětlení informace, které se daly předpokládat i podle výsledků permutation feature importance a to, že nejdůležitějšími proměnnými pro predikci této instance jsou proměnné Title a Sex, ale přidává informaci o tom jakou hrají v této predikci dané hodnoty (Mr a muž). Nicméně mírně překvapivým může být role jakou hraje proměnná Parch, kdy fakt, že cestující cestoval sám, zvyšuje šanci na přežití. Zatímco Fare v tomto případě, na rozdíl od toho, co tvrdí váhy z permutation feature importance, nehraje velkou roli.

Pro srovnání vysvětlení druhé instance, na obrázku 5.8, kdy predikce modelu říká naopak, že cestující přežil, respektive přežila, jelikož se jedná o ženu, ukazuje poměrně velký negativní vliv vyššího věku. To je něco, co bychom čistě na základě permutation feature importance nemuseli očekávat. Nicméně



Obrázek 5.7: LIME vysvětlení pro 1. instanci jako *pyplot* graf.



Obrázek 5.8: LIME vysvětlení pro 2. instanci.

negativní vliv odpovídá očekávání na základě PD grafu proměnné Age, který sice v práci neuvádím, ale není problém jej získat z přiloženého Jupyter notebooku.

5.1.6 Anchors

Pro získání anchor pravidel jsem použil implementaci v knihovně alibi[38]. Vysvětlovat budu opět instance uvedené v tabulce 5.1.

Pravidlo založené na první instanci může pro precision threshold $0,95$ vypadat například následovně (opět se jedná o nedeterministickou metodu, takže výsledky se mohou lišit):

```
IF Title = Mr
AND Fare <= 7.00
THEN Survived = 0
WITH Precision: 0.99
AND Coverage: 0.21
```

Nicméně pokud snížíme precision threshold na hodnotu $0,75$ dostaneme ještě jednodušší pravidlo, na kterém lze pěkně ilustrovat zajímavou vlastnost, a dost možná nedokonalost, modelu:

```
IF Title = Mr
THEN Survived = 0
WITH Precision: 0.88
AND Coverage: 0.57
```

Toto pravidlo vlastně říká, že pro 88% všech instancí s `Title = Mr` predikuje model, že cestující nepřežil. To nemusí být nutně nesprávně. Abychom mohli tvrdit, že je toto chování nechtěné, museli bychom podrobněji prozkoumat data, závislosti v nich a porovnat je s realitou. Nicméně se určitě jedná o vlastnost, na kterou je dobré si dát pozor a pokud bychom model vyvíjeli, tak ověřit, proč tomu tak je.

Pro druhou instanci vypadá pravidlo s precision threshold $0,95$ už příliš konkrétně a má tedy malou coverage:

```
IF Title = Mrs
AND Pclass <= 2.00
AND Sex = female
AND Fare > 31.00
AND Parch <= 0.00
THEN Survived = 1
WITH Precision: 0.98
AND Coverage: 0.03
```

Snížením precision threshold na hodnotu 0,85 už dostaneme použitelnější pravidlo:

```
IF Title = Mrs
AND Pclass <= 2.00
THEN Survived = 1
WITH Precision: 0.88
AND Coverage: 0.09
```

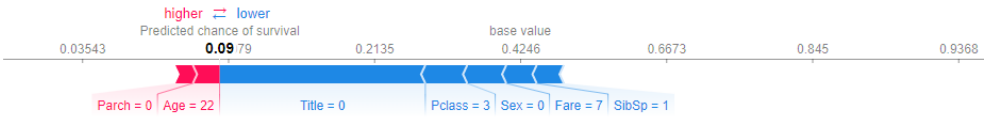
Dalším snižováním precision threshold se pak dostaneme k podobnému pravidlu jako u první instance, jenom pro titul Ms:

```
IF Title = Mrs
THEN Survived = 1
WITH Precision: 0.79
AND Coverage: 0.14
```

5.1.7 SHAP

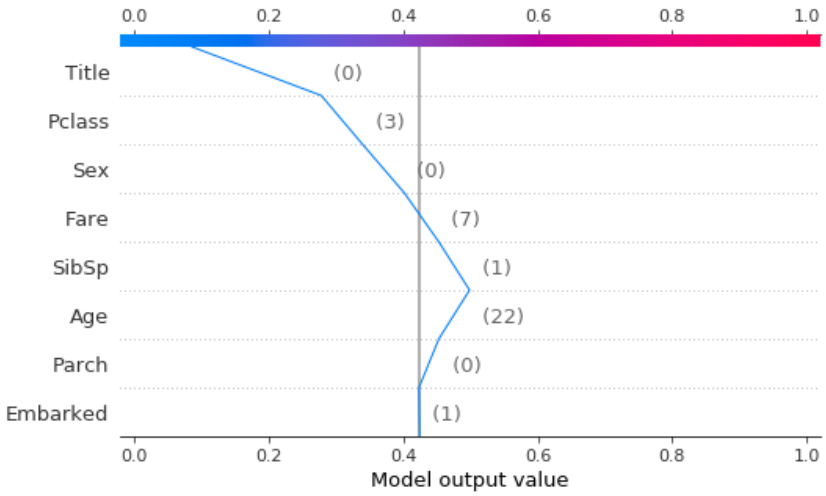
Implementaci metody SHAP poskytuje knihovna shap[44] zmíněná výše. Základním grafem, který nabízí je tzv. *force* graf, viz obrázek 5.9, který symbolizuje jednotlivé proměnné jako „síly“ působící na základní hodnotu predikce (označeno v grafu jako „base value“). Další možnou vizualizací stejných hodnot je *decision* graf na obrázku 5.10, tento graf je vhodné číst „zespoda“, tedy od nejméně důležité proměnné (podle její Shapleyho hodnoty) a to tak, že predikce začíná na svojí základní hodnotě na spodní ose grafu a postupně přidáváním proměnných se posouvá v určitém směru, podle její Shapleyho hodnoty, což symbolizuje modrá čára, která na horní ose končí v hodnotě predikce modelu.

Knihovna shap ale trpí podobným nedostatkem jako PDPbox a to nemožností pojmenovat hodnoty kategorické proměnné. Kvůli tomu nemusí být grafy tak přehledné a lidsky čitelné. Příklad takové proměnné je třeba v obrázku 5.9 proměnná Title, kdy $Title = 0$ není ideální a pro interpretaci tohoto grafu musíme vědět, co tato hodnota znamená.



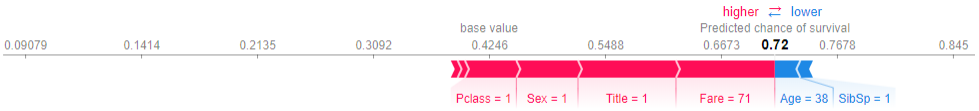
Obrázek 5.9: SHAP force graf pro 1. instanci.

Interpretace pomocí SHAP se v tomto případě mírně odlišuje od interpretace poskytnuté metodou LIME. Obě metody se shodují v důležitosti proměnné Title a ve směru, kterým jednotlivé proměnné ovlivňují predikci, ale v jejich důležitosti nejsou vždy za jedno. Nejvýraznější rozdíl je, že SHAP dává poměrně výrazně vyšší vliv proměnné Age než LIME.



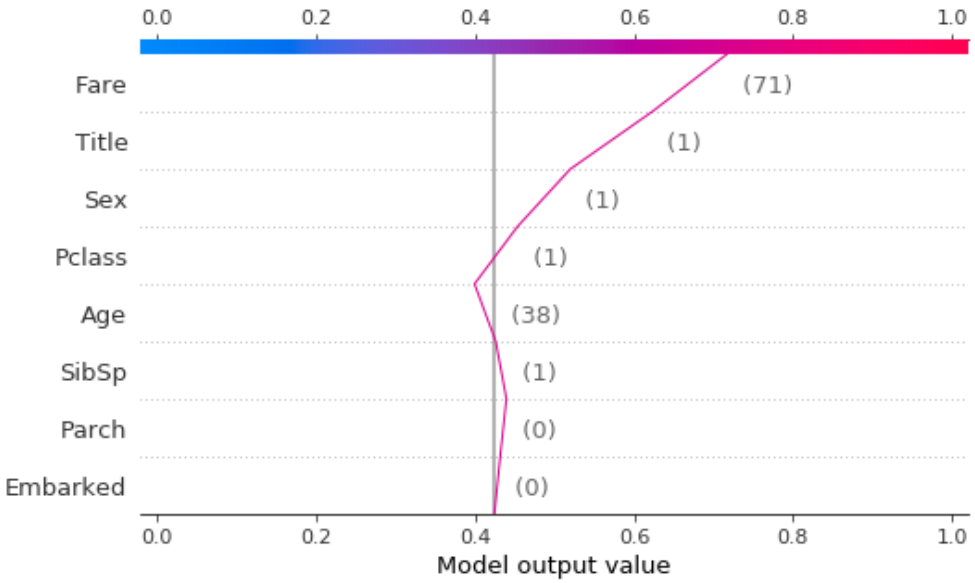
Obrázek 5.10: SHAP decision graf pro 1. instanci.

U druhé instance, viz obrázky 5.11 a 5.12, už ale pozorujeme zásadnější rozdíl. Proměnné Fare, které LIME nepřiřazuje nijak vysokou důležitost, přisoudil SHAP dokonce vyšší vliv než proměnné Title. Drobné odchylky ve výsledcích mohou opět být dány nedeterminističností metody, ale také tím, že prvním krokem, který jsem udělal, je vytvoření reprezentantů celkové množiny pomocí algoritmu KMeans, do 100 vážených vzorků. To je nicméně doporučené, kvůli výpočetní náročnosti SHAP a nemělo by to mít příliš velký vliv na výsledky a přesnost metody.



Obrázek 5.11: SHAP force graf pro 2. instanci.

V případě, kdy se metody neshodují, musíme zvolit důvěryhodnější interpretaci na základě jiných dostupných informací, ostatních instancí nebo vlastního instinktu. Osobně bych se tentokráte zatím klonil spíše k výsledku LIME, protože je konzistentnější s objevenou feature importance (ať už permutation feature importance, tak SHAP feature importance) a navíc interpretace, že Title je opravdu hlavní a nejdůležitější proměnnou, přičemž Pclass hraje výrazně nižší roli platí i pro většinu ostatních instancí, ať už u metody LIME, Anchors nebo i SHAP. Jenomže v další části, při hledání protipříkladů, si ukážeme, že pravda by mohla dost dobře být někde uprostřed, nebo i na straně SHAP.



Obrázek 5.12: SHAP decision graf pro 2. instanci.

Pravděpodobnou příčinou „neshody“ těchto dvou metod může být diskretizace, kterou LIME provádí (LIME hodnotí vliv Fare > 31) nebo podobný princip jako popisuji v části 7.2.5.

5.1.8 Counterfactuals

Counterfactuals, tedy protipříklady, implementuje knihovna alibi[38], kterou jsem použil i pro metodu Anchors. Protože dataset obsahuje kategorické proměnné, bylo potřeba použít metodu hledání protipříkladů za použití prototypů (counterfactuals guided with prototypes)[32]. Další výzvou při použití protipříkladů jsou diskrétní hodnoty, např. proměnné, u kterých dávají smysl pouze celočíselné hodnoty. V případě datasetu Titanic se jedná například o proměnné Parch nebo SibSp. Pokud bychom hledání nijak neomezili, pak by protipříklad mohl obsahovat nesmyslné hodnoty a nebyl by tedy užitečný. Nakonec jsem se tedy rozhodl označit všechny proměnné za kategorické s kategorií pro každou unikátní hodnotu v trénovacích datech.

Nalezeným protipříkladem pro první instanci je instance popsaná v tabulce 5.2. Vidíme, že jedna změna, která je sama o sobě dostatečná pro obrácení predikce, je změna titulu z Mr na Col. To souhlasí s naší dosavadní představou o tom, že model se velmi, skoro až výhradně, zaměřuje právě na tuto proměnnou.

Proměnná	**Původní instance**	**Protipříklad**
Pclass	3	3
Sex	0 (muž)	0 (muž)
Age	22	22
SibSp	1	1
ParCh	0	0
Fare	7	7
Embarked	1 (Southampton)	1 (Southampton)
Title	0 (Mr)	13 (Col)
Predikovaná pravděpodobnost přežití	0,086	0,665

Tabulka 5.2: Protipříklad pro 1. instanci.

Protipříklad pro druhou instanci, viz tabulka 5.3, je ovšem zajímavější. V tomto případě totiž nemění Titul, ale 2 jiné proměnné a to Age, z původních 38 let na 67, a Fare, ze 71 na 25. Změna těchto dvou hodnot odpovídá spíše tomu, jak predikci pro danou instanci vysvětlovala metoda SHAP, jelikož se jedná o proměnné s největším pozitivním vlivem a největším negativním vlivem na predikci. Naopak mírně zpochybňuje LIME, která Fare nepovažovala za příliš důležitou pro predikci v tomto případě.

Proměnná	**Původní instance**	**Protipříklad**
Pclass	1	1
Sex	1 (žena)	1 (žena)
Age	38	67
SibSp	1	1
ParCh	0	0
Fare	71	25
Embarked	0 (Cherbourgh)	0 (Cherbourgh)
Title	1 (Mrs)	1 (Mrs)
Predikovaná pravděpodobnost přežití	0,717	0,346

Tabulka 5.3: Protipříklad pro 2. instanci.

Nesmíme však zapomínat, že se jedná pouze o odhady a dohady, odvozené závěry nejsou stoprocentní a pravda může být i někde jinde. Bohužel však v případě interpretace modelů nemáme lepší možnost, než vytvářet právě takové, trochu nejisté, závěry. Přesto ale platí, že v případě tohoto modelu jsme získali obecnou představu o tom, jak se chová, v čem je jeho hlavní nedokonalost (přílišné spoléhání na proměnnou Title) a nějakým způsobem jsme vysvětlili predikce ve dvou konkrétních případech.

5.2 California Housing

5.2.1 Data

Jako další příklad interpretace modelu jsem zvolil regresní problém odhadu ceny Kalifornských nemovitostí, respektive bloků nemovitostí[48]. A to proto, abych ilustroval odlišnosti, a některé obtíže, interpretace regresních modelů.

Všechny proměnné jsou spojité a jejich význam je popsán v následující tabulce:

Proměnná	**Význam**
longitude	Zeměpisná délka
latitude	Zeměpisná šířka
housingMedianAge	Medián stáří nemovitosti v bloku
totalRooms	Celkový počet pokojů v bloku
totalBedrooms	Celkový počet ložnic v bloku
population	Celkový počet domácností v bloku
medianIncome	Medián ročního příjmu domácností v 10000$

5.2.2 Model

Po drobném předzpracování dat, tentokráte pouze normalizace (přeškálování) hodnot pomocí `StandardScaler()` z knihovny `sklearn.preprocessing` a úpravě výstupních hodnot, tak aby obsahovaly rozdíl mezi průměrnou hodnotou v tisících, jsem natrénoval podobný model, tedy neuronovou síť, jako v ukázce s datasetem Titanic:

```
MLPRegressor(solver='lbfgs', alpha=1e-5,
↪  hidden_layer_sizes=(8, 3), random_state=42)
```

R^2 (koeficient determinace) tohoto modelu je přibližně $0,737$.

5.2.3 Feature importance

Na rozdíl od předchozího případu můžeme použít permutation feature importance na testovací i trénovací data. Pro trénovací data vypadá výsledná tabulka následovně:

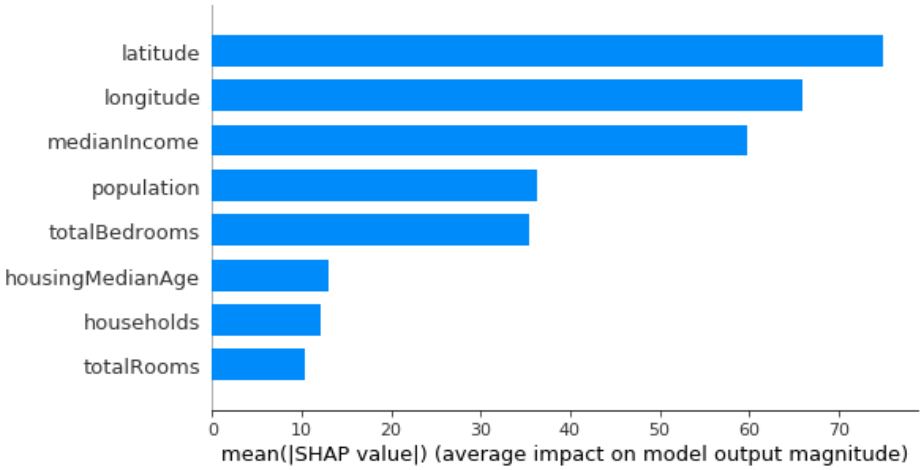
Váha	Proměnná
1.4106 ± 0.0208	latitude
1.2080 ± 0.0360	longitude
0.9484 ± 0.0241	medianIncome
0.5457 ± 0.0143	totalBedrooms
0.4297 ± 0.0113	population
0.1353 ± 0.0069	totalRooms
0.0779 ± 0.0041	housingMedianAge
0.0674 ± 0.0036	households

A pro testovací data:

Váha	Proměnná
1.4130 ± 0.0607	latitude
1.1874 ± 0.0558	longitude
0.9405 ± 0.0569	medianIncome
0.5436 ± 0.0128	totalBedrooms
0.4304 ± 0.0125	population
0.1289 ± 0.0077	totalRooms
0.0822 ± 0.0084	housingMedianAge
0.0646 ± 0.0028	households

V obou případech jsou jak váhy, tak i pořadí proměnných téměř stejné. To značí, že naše trénovací a testovací data jsou podobná a tedy rozdělení původních dat do těchto množin proběhlo správně.

Feature importance pomocí metody SHAP se opět příliš neliší od permutation feature importance.

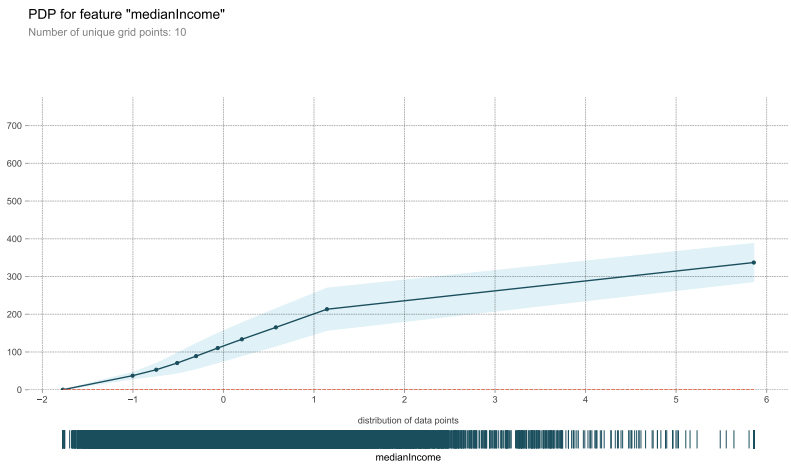


Obrázek 5.13: Feature importance pomocí metody SHAP.

Ukazuje se, že tento model zakládá svoji predikci zejména na geolokačních údajích, tedy zeměpisné šířce a délce, ale také na střední hodnotě příjmu v daném bloku. Tato kombinace zní jako rozumný základ pro predikci hodnoty nemovitosti.

5.2.4 Partial dependence grafy

Partial dependence grafy a jejich konstrukce se v případě regrese nijak významně neliší od předchozího případu klasifikace. Částečně je tomu tak i proto, že pro daný model byla k dispozici metoda vracející pravděpodobnost, že instance patří do dané třídy. To není u klasifikačních modelů nijak neobvyklé a je vhodné analyzovat právě tyto predikované pravděpodobnosti.



Obrázek 5.14: Partial dependence pro proměnnou medianIncome

Na obrázku 5.14 můžeme vidět vliv proměnné medianIncome na výstup. Podle očekávání s rostoucím příjmem v oblasti roste i predikovaná hodnota nemovitosti. Opět je ale potřeba dát si pozor na distribuci hodnot a zda neděláme závěry nad nějakou částí grafu, pro kterou nemáme v datech žádné instance.

Grafy pro ostatní proměnné a také *actual plot* lze v případě zájmu vygenerovat v přiloženém Jupyter notebooku *Housing.ipynb*.

5.2.5 LIME

5.2.5.1 Instance

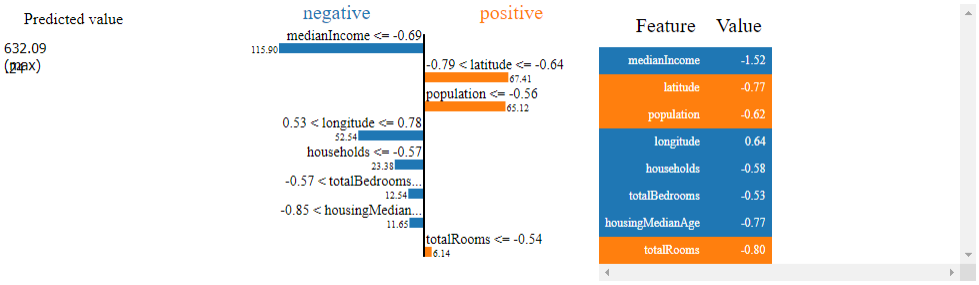
Tentokráte jsem se zaměřil pouze na jednu instanci, na které ilustruji využití stejných metod jako v případě datasetu Titanic. V tabulce 5.4 uvádím jak původní hodnoty proměnných, tak i normalizované hodnoty, které jsou ve skutečnosti používané pro predikci.

proměnná	Zkoumaná instance	Normalizované hodnoty
longitude	-118.28	0.644
latitude	33.98	-0.773
housingMedianAge	19	-0.766
totalRooms	883	-0.803
totalBedrooms	313	-0.534
population	726	-0.618
households	277	-0.582
medianIncome	0.9809	-1.521

Tabulka 5.4: Zkoumaná instance datasetu California Housing.

5.2.5.2 Výsledky metody

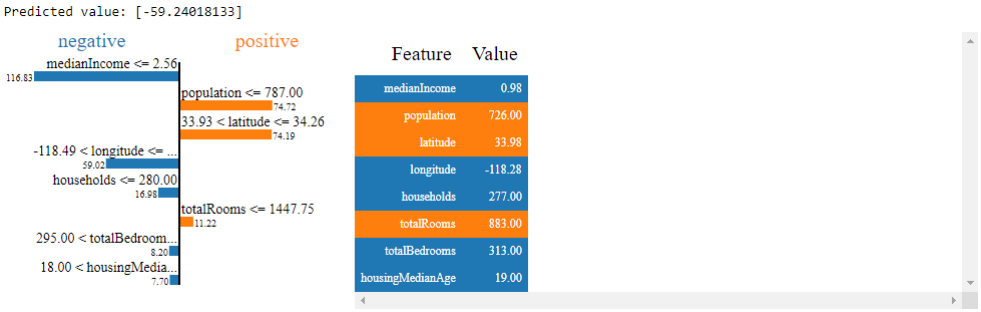
Knihovna lime[43] podporuje nejen klasifikační problémy, ale i regresní, jenomže pro regresi s podobnými hodnotami výstupu si neporadí se zobrazením „Predicted value“ zrovna dobře, viz obrázek 5.15.



Obrázek 5.15: Výstup metody LIME pro zkoumanou instanci s normalizovanými daty.

Další nevýhodou je, že pokud použijeme nijak neobalenou metodu, jejíž vstupní hodnoty jsou již normalizované, není výstup lidsky přívětivý. Proto je v tomto případě lepší vytvořit tzv. *wrapper* funkci, tedy funkci, která obalí volání modelu tak, že když ji zavoláme na instance s původními hodnotami, vrátí predikci modelu pro instanci s příslušnými normalizovanými hodnotami. LIME pak prozkoumává prostor s instancemi s původními hodnotami a

vysvětlení se tedy pohybuje nad tímto lidsky přívětivějším prostorem, to ilustruje obrázek 5.16. Použitá wrapper funkce je v tomto případě jednoduchá, viz ukázka 5.17



Obrázek 5.16: Výstup metody LIME pro zkoumanou instanci s původními daty.

```
from sklearn import preprocessing.StandardScaler
standard_scaler = StandardScaler()
standard_scaler.fit(X) # X is california_housing dataset
...
def wrap_orig_vals(instance):
    return clf.predict(standard_scaler.transform(instance))
```

Ukázka 5.17: Použitá wrapper funkce pro LIME na obrázku 5.16.

Z výstupů můžeme vidět, že hlavním důvodem, proč model pro tento případ predikuje nižší hodnotu nemovitosti je nízký příjem obyvatel žijících v dané lokalitě. Opačný vliv má ale fakt, že v oblasti nebydlí mnoho lidí a také geolokační umístění (součet vlivu zeměpisné délky a šířky) má nakonec, alespoň podle výstupu LIME, mírně pozitivní vliv na výsledek.

5.2.6 Anchors

Použití Anchors pro regresní problémy je už ale na rozdíl od předchozích metod problematické. Prakticky totiž není téměř možné získat dostatečnou, pokud se navíc pohybujeme ve opravdu spojitém prostoru výstupů, tak vůbec jakoukoli, precision. Ukázka 5.18 ilustruje neschopnost nalézt pravidlo pro čistě regresní problém.

Abychom byli schopni nalézt použitelná pravidla, je vhodné výstup diskretizovat, čímž vlastně problém převedeme na klasifikaci. Čím menší granularitu bude výstup mít, tím snazší a lepší pravidla budeme schopni nalézt. Pro ukázku jsem zvolil nejmenší možnou granularitu, tedy rozdělení do dvou tříd – predikovaná hodnota je větší než (nebo rovno) průměrná a predikovaná

```
IF medianIncome <= -0.69
 AND -0.57 < totalBedrooms <= -0.24
 AND 0.53 < longitude <= 0.78
 AND households <= -0.57
 AND -0.79 < latitude <= -0.64
 AND -0.85 < housingMedianAge <= 0.03
 AND totalRooms <= -0.54
 AND population <= -0.56
THEN Value = -59.24018133470079
WITH Precision: 0.00
AND Coverage: 0.00
```

Ukázka 5.18: Anchors pravidlo pro původní funkci.

hodnota je menší než průměrná. Určitě by ale bylo možné diskretizovat výstup mnoha různými způsoby.

Při použití wrapper funkce 5.19 můžeme pomocí Anchors získat pro zkoumanou instanci například následující pravidlo:

```
IF medianIncome <= 2.56
THEN Value = -1
WITH Precision: 0.97
AND Coverage: 0.25
```

Nebo pokud zvýšíme precision threshold:

```
IF medianIncome <= 2.56
  AND totalBedrooms <= 435.00
  AND housingMedianAge <= 37.00
THEN Value = -1
WITH Precision: 1.00
AND Coverage: 0.08
```

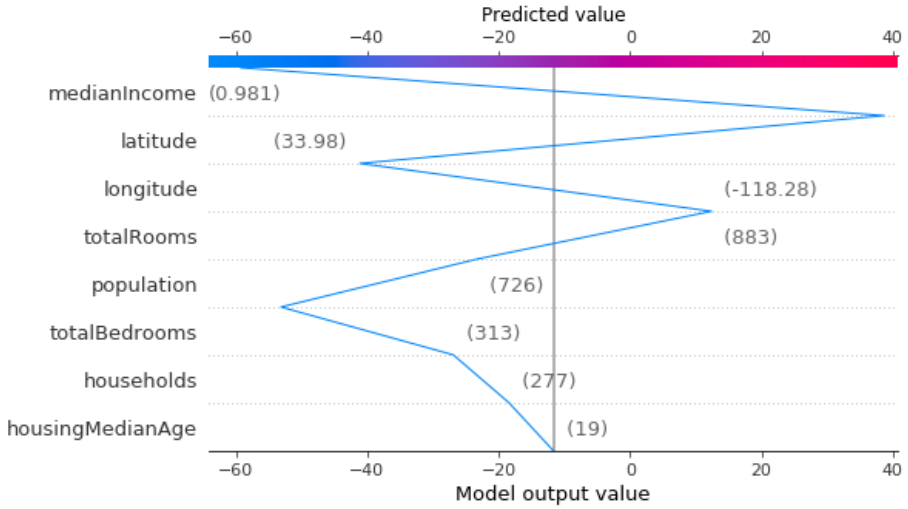
```python
def wrap_discretize(instances):
  res = []
  for x in clf.predict(standard_scaler.transform(instances)):
    if x < 0:
      res.append(-1)
    else:
      res.append(1)
  return np.array(res)
```

Ukázka 5.19: Použitá wrapper funkce pro Anchors.

5.2.7 SHAP

SHAP je metoda, která naopak počítá s regresním výstupem, i v případě klasifikace pracuje lépe s pravděpodobnostmi, že proměnná patří do určité třídy. Použití SHAP je tedy pro regresi bezproblémové.

Obrázek 5.20: SHAP force plot pro zkoumanou instanci.



Obrázek 5.21: SHAP decision plot pro zkoumanou instanci.

Na obrázcích 5.20 a 5.21 můžeme vidět *force* a *decision* grafy pro zkoumanou instanci. Podobně jako v případě předchozího datasetu se vysvětlení trochu liší. Největší rozdíl je v posouzení vlivu proměnné *population*, které LIME přiřazuje o poznání větší vliv než SHAP. LIME opět více odpovídá feature importance získané jak pomocí permutation feature importance tak pomocí průměrné absolutní SHAP hodnoty. Na první pohled bychom se tedy znovu mohli přiklánět spíše k vysvětlení, které poskytuje LIME, nicméně jak už jsme si ukázali v předchozím případě, nemusela by to nutně být zcela správná volba.

5.2.8 Counterfactuals

Counterfactuals z podstaty věci vyžadují klasifikační problém. Bohužel nestačí jenom snadno výstup regresního modelu diskretizovat, je potřeba jednotlivým třídám přiřadit pravděpodobnosti (podobně jako metoda `predict_proba()`

modelů knihovny *sklearn*), aby byla metoda schopna prohledávat prostor, tj. aby věděla, zda se změnou hodnoty proměnné přiblížila k protipříkladu nebo naopak. Toho lze dosáhnout například přeškálováním hodnot predikce pro celý dataset do intervalu $[-1, 1]$, ze kterého pak vytvoříme „pravděpodobnosti“ pro třídy „menší než průměrná hodnota“ a „větší než průměrná hodnota“, viz ukázka 5.22.

```
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
scaler.fit(wrap_orig_vals(orig).reshape(-1, 1))
def wrap_cf(instances):
  res = []
  for x in clf.predict(standard_scaler.transform(instances)):
    transformed = scaler.transform(x.reshape(-1, 1))[0][0]
    is_positive = 0.5 + transformed/2
    res.append([1 - is_positive, is_positive])
  return np.array(res))
```

Ukázka 5.22: Funkce pro přeškálování výsledků regrese do pravděpodobnosti tříd „menší než průměrná hodnota“ a „větší nebo rovno průměrné hodnotě“. Pro definici *standard_scaler* viz 5.17.

Metoda counterfactuals použitá na predikci obalenou wrapper funkcí 5.22 může vytvořit například následující protipříklad:

proměnná	**Zkoumaná instance**	**Protipříklad**
longitude	-118.28	-118.932
latitude	33.98	33.264
housingMedianAge	19	19
totalRooms	883	883
totalBedrooms	313	313
population	726	762
households	277	277
medianIncome	0.9809	1.052

Tabulka 5.5: Protipříklad pro instanci 5.4.

Predikce pro původní instanci byla $-59, 24$, tedy méně než průměr a predikce pro protipříklad $0, 34$, tedy více než průměr. Změny odpovídají tomu, co tvrdí feature importance a ostatní metody, jelikož proměnné které bylo nutné upravit jsou zeměpisná šířka a délka a příjem v oblasti.

Kapitola 6

ExpyBox

6.1 Úvod

Na předchozích případech jsem ilustroval, že interpretace modelu vyžaduje práci s řadou různých knihoven pro různé metody interpretability modelů, z nichž mnohé mají řadu parametrů, nebo hyperparametrů, které nelze snadno optimalizovat programaticky, jelikož na rozdíl od například trénování modelů, nemáme k dispozici žádnou objektivní metriku, kterou bychom chtěli maximalizovat. Cílem interpretace je totiž především vysvětlit model srozumitelně pro člověka a bez lidského ohodnocení tedy můžeme jen těžko určit, zda je interpretace „lepší".

Pro usnadnění práce a jednodušší experimentování s hodnotami parametrů jsem se rozhodl vytvořit knihovnu, která umožní uživateli jednoduché vytvoření interaktivního *Jupyter notebooku* pro různé metody interpretability.

6.2 Technologie

6.2.1 Jupyter notebook

Technologie Jupyter notebooků (dokumentů) je pro většinu datových analytiků velmi dobře známá a svým způsobem přirozená. Jedná se vlastně o dokumenty, které mohou obsahovat jak spustitelný kód, tak textové prvky ve značkovacím formátu markdown. Jupyter může podporovat řadu jazyků[49], ale jeho název vznikl kombinací JUlia, PYThon a R.

Pro interpretabilitu ale i data exploration (prozkoumávání dat) a další činnosti datových analytiků se jedná o velmi praktický nástroj, jelikož umožňuje rychlé a opakované spouštění úseků kódu, jehož výstup (např. graf) okamžitě zobrazí. Navíc lze vytvořit i interaktivní notebooky, které pak poskytují analytikovi, ale případně i samotnému zákazníkovi, jednoduché grafické rozhraní s možností úpravy kódu.

6.2.2 ipywidgets

ipywidgets[50] je knihovna poskytující interaktivní HTML „widgety“ (např. dropdown nebo textová pole) pro Jupyter notebooky. Umožňuje tedy vytvářet interaktivní notebook, kde uživatel zadává data do formulářů a Jupyter kernel – program, se kterým notebook komunikuje a který má na starosti provádění kódu – s nimi nějakým způsobem pracuje.

6.3 Použití knihovny

Knihovna je vytvořená pro použití v prostředí Jupyter notebooku, prvním krokem je tedy takový notebook otevřít a připravit si v něm model a data, která budeme používat. Poté stačí naimportovat třídu `ExpyBox` a vytvořit její instanci:

```
from expybox import ExpyBox
expybox = ExpyBox(train_data, predict_function, ...)
```

přičemž je doporučeno vyplnit všechny aplikovatelné parametry. Seznam parametrů i s jejich popisem lze dohledat v dokumentaci zveřejněné na stránce *Read the Docs*[51], na adrese `https://expybox.readthedocs.io/en/latest/`.

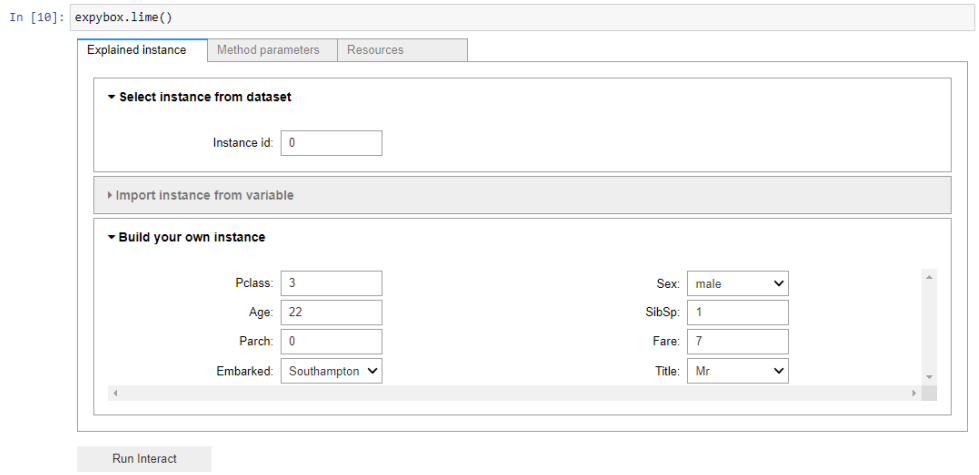
Formuláře pro jednotlivé podporované metody pak uživatel vytvoří jednoduchými voláními metod třídy `ExpyBox`, například:

```
expybox.lime()
```

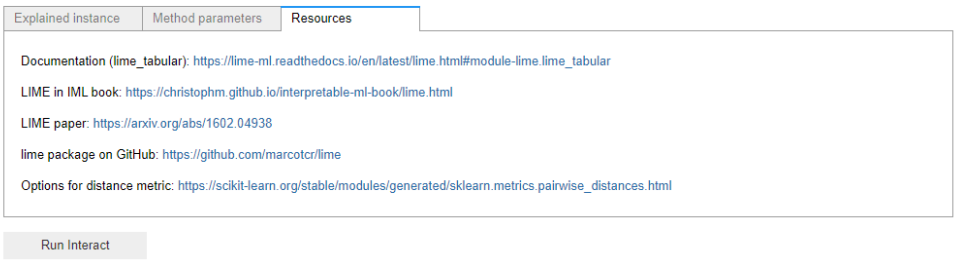
vytvoří formulář na obrázku 6.1. V záložce „Explained instance“ si uživatel zvolí vysvětlovanou instanci a v záložce „Method parameters“ požadované parametry metody. Záložka „Resources“ obsahuje seznam užitečných odkazů pro danou metodu, viz obrázek 6.2. Po stisknutí tlačítka „Run Interact“ se metoda spustí a její výsledek se zobrazí přímo v notebooku.

Knihovna zatím podporuje následující metody:

- Partial dependence plot
- LIME
- Anchors
- SHAP
- SHAP feature importance



Obrázek 6.1: Formulář pro metodu LIME.



Obrázek 6.2: Záložka „Resources“ pro metodu LIME.

6.4 Implementace

6.4.1 Návrh

Samotnou knihovnu jsem se snažil koncipovat tak, aby nebylo náročné přidávat další podporované metody. Proto se knihovna skládá vlastně ze dvou částí: třídy `ExpyBox` a tříd typu `Explainer` (potomků třídy `Explainer`).

6.4.1.1 Třída ExpyBox

Třída ExpyBox je hlavní třída, kterou uživatel instancuje. Ta pak drží uživatelem zadaná data, jako predict funkci zkoumaného modelu nebo trénovací data, a zajišťuje správná volání jednotlivých `Explainer` tříd. Také implementuje tvorbu opakovaně používaných záložek ve formulářích – volbu vysvětlované instance a záložku „Resources“.

Tato třída se tedy stará o vytvoření formuláře, naplnění výše zmíněných záložek pokud jsou třeba (globální metody vysvětlují celý model, nepotřebují tedy žádnou vysvětlovanou instanci) a správnou instanciaci příslušné třídy

`Explainer`. Po odeslání formuláře, tj. když co uživatel klikne na tlačítko „Run Interact“, se také stará o extrakci hodnot z formuláře, případnou tvorbu vysvětlované instance a poté volání metody `explain` příslušné instance třídy `Explainer` se správnými parametry.

6.4.1.2 Explainer

Každá třída typu `Explainer` implementuje jednu metodu interpretability. Metoda `build_options` je zodpovědná za tvorbu formuláře, který se zobrazí v záložce „Method parameters“ kde si uživatel nastaví jednotlivé parametry metody. Dále musí poskytovat metodu `explain`, která provede implementovanou metodu s příslušnými parametry a její výsledek zobrazí v notebooku.

Dále definuje dictionary `resources`, který vždy obsahuje popisek odkazu jako klíč a odkaz jako hodnotu (obojí ve formátu jednoduchého řetězce), na základě kterého `ExpyBox` konstruuje záložku „Resources“, a booleovský příznak `require_instance`, který říká, zda metoda explain vyžaduje konkrétní instanci, kterou bude vysvětlovat. Díky tomuto parametru také `ExpyBox` ví, zda je potřeba vytvořit i záložku „Explained instance“.

```
class Explainer:
  resources = {}
  require_instance = False

  def build_options(self) -> Tuple[Dict[str, Widget], Box]:
    pass

  def explain(self, options: Dict[str, Any], instance:
  ↪  Optional[array] = None) -> None:
    pass
```

Ukázka 6.3: Definice třídy Explainer (bez *docstring* dokumentace), která slouží jako základ pro třídy definující jednotlivé metody.

6.4.2 Přidání nové metody

Jak jsem již zmínil výše, hlavním cílem návrhu bylo, aby přidání dalších metod probíhalo snadno, bez větších zásahů do implementace. Ve stávajícím návrhu knihovny stačí pro přidání další metody naimplementovat podtřídu `Explainer`, která implementuje obě metody zmíněné výše – `build_options` a `explain`. Poté už stačí jen ve třídě `ExpyBox` vytvořit metodu, kterou bude moci uživatel zavolat. Tato metoda pouze vytvoří instanci dané podtřídy `Explainer` a zavolá svou třídní metodu `_display_interact`, podobně jako v ukázce 6.4.

```
def pdplot(self) -> None:
  """
  Create dialog for partial dependence plot
  :return: None
  """
  pdp = PDP(train_data=self.X_train,
            predict_function=self.predict_function,
            feature_names=self.feature_names,
            globals_options=self.kernel_globals.keys())
  self._display_interact(explainer=pdp,
  ↪  explain_instance=pdp.require_instance)
```

Ukázka 6.4: Metoda `pdplot` třídy `ExpyBox`.

6.4.3 Proměnné kernelu

Pro některé parametry metod interpretability (jako například custom grid points pro partial dependence plot) je vhodné, aby knihovna, resp. funkce, která samotnou metodu volá, měla přístup k proměnným definovaným uživatelem na úrovni notebooku, tedy v kernelu. K těmto proměnným samozřejmě ale knihovna přístup nemá a tak je potřeba nějak jí je předat.

Nejjednodušší možností by bylo předávat vždy danou konkrétní proměnnou jako parametr funkce, tj. z volání `ExpyBox(...).pdplot()` by se stalo něco jako `ExpyBox(...).pdplot(cust_grid_points=variable)`. To ale neodpovídá myšlence knihovny, že by vždy mělo stačit pouze jedno volání metody jako je `pdplot()` i pro opakované experimenty, při kterých si uživatel jen navolí jiné parametry.

Lepší by tedy bylo, kdyby uživatel mohl do určitého textového pole vepsat název proměnné, kterou by si poté knihovna sama dohledala. Jak už jsem ale zmínil výše, k proměnným kernelu nemá knihovna jen tak přístup. V jazyce Python však existuje globální slovník (dictionary), který obsahuje všechna jména přístupná z kontextu a tedy i všechny proměnné definované uživatelem. Tento slovník je přístupný pomocí příkazu `globals()` a pokud jej uživatel předá při instanciaci třídy `ExpyBox()`, pak bude jeho pomocí třída mít, díky vlastnosti mělké kopie, přístup k proměnným kernelu. A to i těm, které uživatel definuje později než ve chvíli, kdy slovník předá.

Takový slovník si uživatel může samozřejmě nadefinovat a udržovat i sám. Nemusí tedy nutně předávat všechna jména na úrovni modulu. Pro srovnání pokud uživatel předá instanci následující slovník jako parametr `kernel_globals`:

```
a = [0, 5, 10]
user_dict = {'a': a}
```

```
b = False
expybox = ExpyBox(..., kernel_globals=user_dict)
```

bude mít `expybox` přístup k proměnné `a`, kterou bude uživateli v případech, kde očekává proměnnou, našeptávat a bude schopna ji dohledat. Proměnnou `b` už ale neuvidí a nebude se tak ani schopna k jejímu obsahu dostat.

Nejjednodušší použití je tedy předání samotného slovníku `globals()`:

```
a = [0, 5, 10]
b = False
expybox = ExpyBox(..., kernel_globals=globals())
c = [10, 11, 12]
```

V tomto případě bude `expybox` mít přístup ke všem proměnným, včetně `c`. O udržování slovníku `globals()` se totiž stará Python sám. Zároveň ale bude mít instance přístup i ke všem dalším proměnným a importovaným jménům, z nichž některé jsou spíše technického charakteru jako například `__name__`.

6.5 Publikace

Knihovnu jsem publikoval na Python Package Index (PyPI.org)[52], takže její instalace probíhá, stejně jako instalace mnoha dalších knihoven, jednoduchým příkazem `pip install expybox`. Bohužel kvůli použití knihovny alibi, kterou nelze používat s 32bitovým Pythonem, selhává instalace závislostí právě při použití 32bitového Pythonu. Doporučeným systémem je tedy Python 3.7 64bit.

Zdrojový kód a dokumentace jsou k dispozici ve veřejném GitHub repozitáři na adrese `https://github.com/Kukuksumusu/expybox`.

Kapitola 7

Vortex

Použití knihovny ExpyBox jsem se rozhodl ilustrovat na credit scoringovém modelu *Vortex*. Tento model jsem zvolil přesto, že se jedná o model, který je sám o sobě plně interpretovatelný – logistickou regresi. Můžeme si na něm totiž ukázat, jak si jednotlivé metody poradí s jeho interpretací, když k němu přistoupíme jako k black-box modelu.

7.1 Model

Jelikož se jedná o citlivý model, nemohu jej v práci zveřejnit přesně tak, jak se používá v praxi. Proto jsem vytvořil jeho upravenou kopii. Stejně tak nemohu z bezpečnostních důvodů uveřejnit názvy a význam proměnných.

Jedná se o model logistické regrese, takže výpočet predikce vypadá následovně:

$$P(y = 1) = 1 - \frac{1}{e^{const + \sum_{i=0}^{7} \beta_i * woe_b}}$$

const značí konstantu, tzv. *intercept* a woe_b je *weight of evidence* pro interval, ve kterém se nachází hodnota proměnné i.

Samotnou implementaci modelu lze nalézt na přiloženém CD v souboru *vortex_anonymized.py*

7.2 Diskretizovaná data

Nejprve si ukážeme, jak si metody poradí pokud data jsou již diskretizovaná do jednotlivých intervalů.

Proměnná	Váha
x0	0.4761476
x1	0.3208511
x2	0.2064462
x3	0.2587311
x4	0.3342483
x5	0.3187723
x6	0.2848142
x7	0.4116576

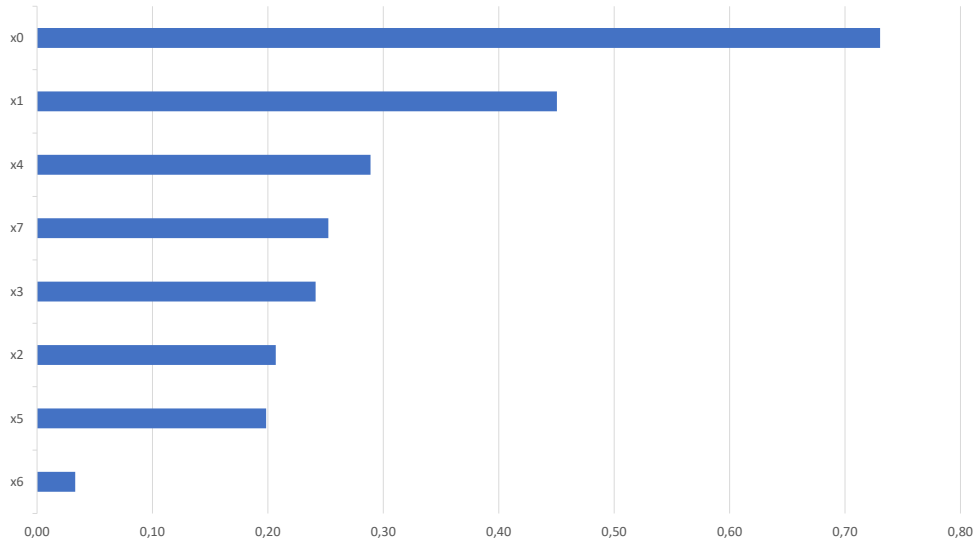
Tabulka 7.1: Váhy (koeficienty β) proměnných v modelu Vortex.

7.2.1 Feature importance

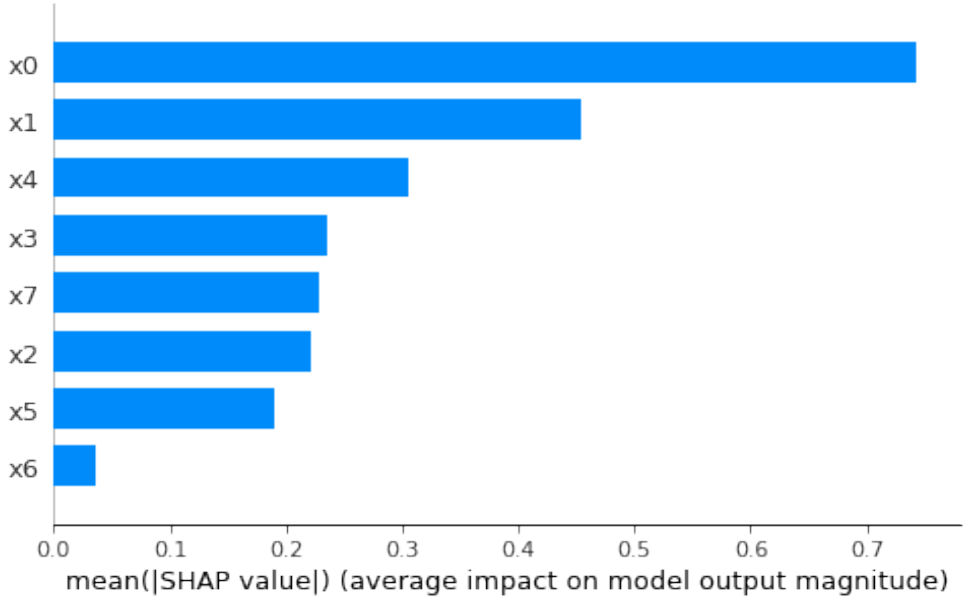
Jelikož známe přesně postup, jakým model tvoří predikce, můžeme vypočítat průměrný vliv proměnné na instanci z trénovacích dat přesně jako

$$V_{x_i} = \frac{\sum_{b=0}^{n} c_{x_b=n} * |woe_b| * \beta}{\text{počet instancí}}$$

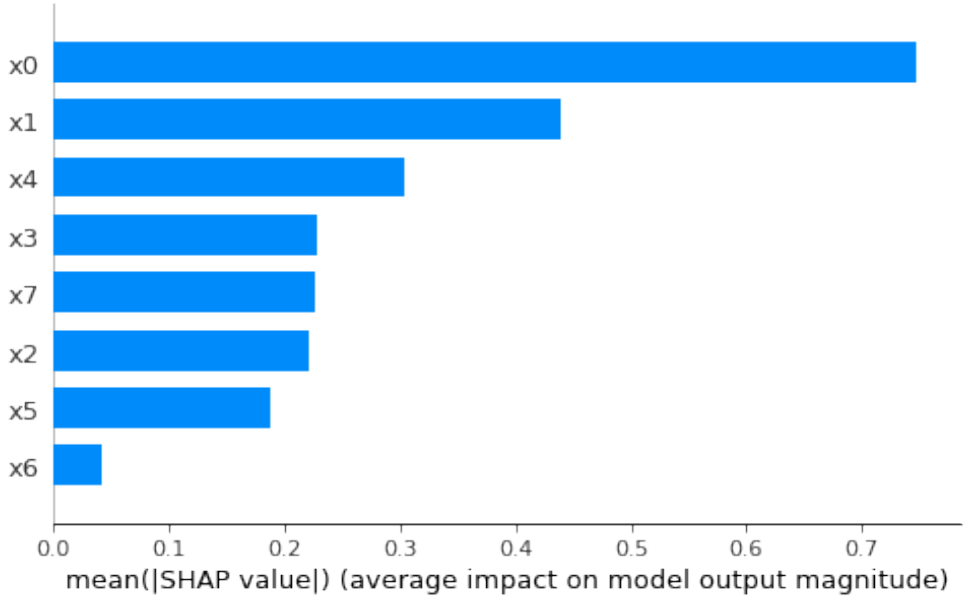
kde n je počet binů a $c_{x_b=n}$ značí počet instancí, pro které hodnota proměnné x_i spadá do binu b. Tímto postupem získáme očekávané Shapleyho hodnoty, viz obrázek 7.1, které můžeme porovnat se skutečným výstupem metody SHAP na obrázku 7.2. Tento výstup jsem získal za použití vzorku 10000 hodnot z původních dat, 100 KMeans centry pro „background data“ (data, ze kterých se vybírají hodnoty pro „chybějící“ proměnné) a také `link='logit'`.



Obrázek 7.1: Očekávaný výstup pro shap feature importance.



Obrázek 7.2: Skutečný výstup shap feature importance.



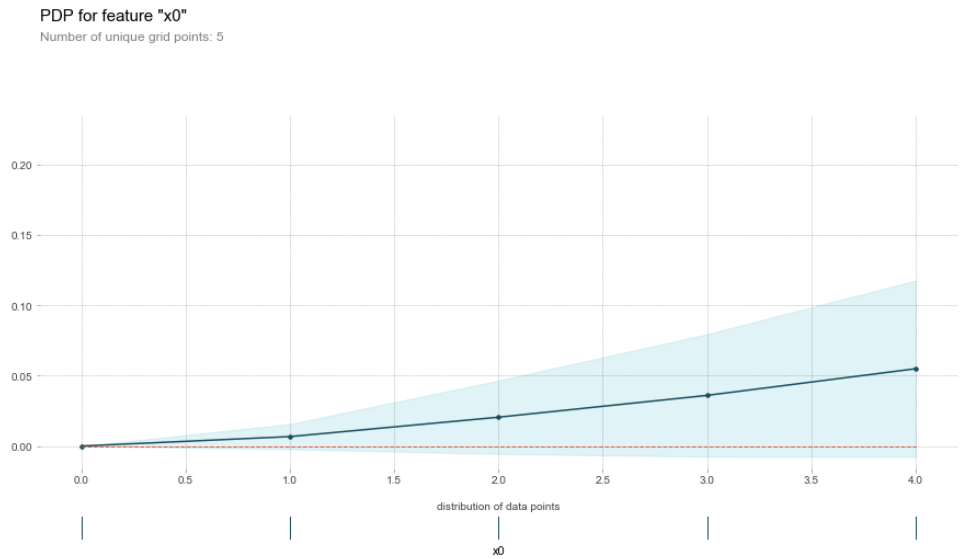
Obrázek 7.3: Výstup shap feature importance pro vzorek 1000 instancí.

Čím vyšší bude počet vzorků a také velikost „background dat“, tím přesněji bude odpovídat výstup očekávání. Nicméně již pro vzorek této velikosti se výstup téměř neliší od skutečnosti. Bohužel ale i pro takto jednoduchý model, navíc dostupný lokálně, trval výpočet přibližně 45 minut. Představa,

že bychom zkoumali model větší (s více proměnnými, složitějším výpočtem apod.), nedej bože dostupným pouze přes internetové api, není úplně příjemná. Pro srovnání na obrázku 7.3 uvádím výstup pro vzorek 1000 instancí, jehož výpočet trval přibližně 5 minut a je v podstatě k nerozeznání.

7.2.2 Partial dependence grafy

Partial dependence grafy budou v tomto případě plně odpovídat očekávání. Proměnná „x0“ se zdá jako nejdůležitější, proto na obrázku 7.4 ukážu její graf. Ten zcela správně odhaluje téměř lineární závislost výstupu na proměnné „x0“.



Obrázek 7.4: Partial dependence graf pro proměnnou „x0“.

7.2.3 LIME

7.2.3.1 Instance

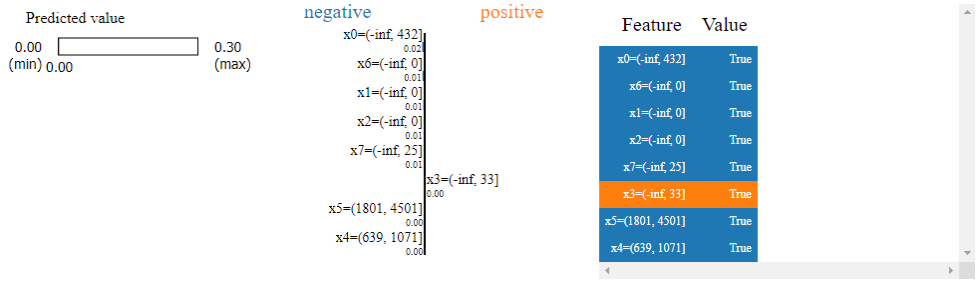
Pro ukázku lokálních metod se zaměřím na instanci popsanou v tabulce 7.2. Sloupec $woe_b * \beta_i$ říká, jaký vliv má proměnná na predikci modelu pro danou instanci.

Proměnná	Hodnota	$\mathbf{woe_b} * \beta_\mathbf{i}$
x0	0	-0.8802
x1	0	-0.6169
x2	0	-0.1997
x3	24	0.1595
x4	650	-0.2580
x5	3251	-0.1411
x6	0	-0.0208
x7	10	-0.3909

Tabulka 7.2: Zkoumaná instance pro model vortex.

7.2.4 Výsledky metody

Bohužel pro tento případ se opět ukazuje nedokonalost zobrazení knihovny LIME, která si dobře neporadí s malými hodnotami predikcí, viz obrázek 7.5. Tento problém lze naštěstí poměrně snadno vyřešit, aby bylo vysvětlení přehlednější. Stačí výstup modelu upravit tak, aby odpovídal pravděpodobnosti v procentech, tj. vynásobit stem.



Obrázek 7.5: LIME bez upravené predict funkce.

Opět jsem tedy vytvořil jednoduchou wrapper funkci:

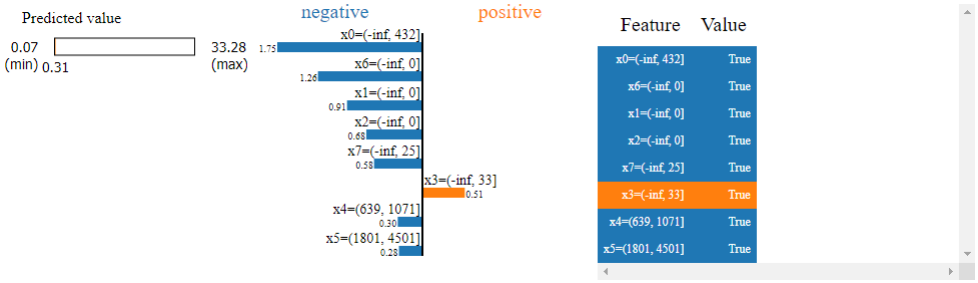
```
def wrap_lime(instances):
  return vortex.predict(instances) * 100
```

Tuto funkci je třeba instanci `ExpyBox` buď přiřadit pomocí

```
expybox.predict_function = wrap_lime
```

nebo vytvořit úplně novou instanci třídy `ExpyBox`, přičemž preferovanější je určitě spíše druhý způsob.

Obrázek 7.6 ukazuje výstup LIME pro zkoumanou instanci s kernel width $0,75 * \sqrt{8}$ a 15000 vzorky pro učení lineárního modelu. Větší počet vzorků jsem zvolil kvůli stabilitě metody. Při použití základní možnosti 5000 vzorků je metoda příliš nestabilní, přičemž čas běhu se téměř nezměnil.



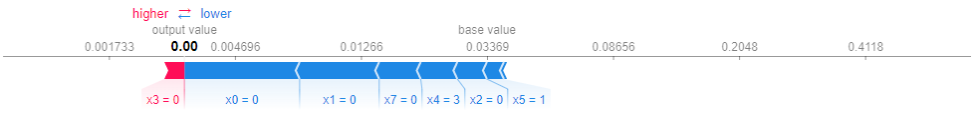
Obrázek 7.6: LIME pro zkoumanou instanci s výstupem v procentech.

Podle očekávání považuje LIME za nejdůležitější vliv proměnné „x0“. U všech proměnných také správně určuje směr, kterým predikci ovlivňují. Za povšimnutí ovšem stojí to, jak velkou váhu LIME přiřazuje proměnné „x6“, která přitom má vliv na predikci v tomto konkrétním případě téměř zanedbatelnou.

Abychom pochopili proč tomu tak je, musíme si nejprve uvědomit, jak k interpretaci LIME přistupuje. LIME natrénuje lineární model na vážených příkladech z okolí vysvětlované instance. To znamená, že vliv proměnné je vlastně rozdíl mezi tím, když má proměnná tuto hodnotu a kdyby měla jinou. A pokud se podíváme na proměnnou „x6“, tak uvidíme, že se jedná o proměnnou s 2 kategoriemi (biny), přičemž hodnota $woe_b * \beta_i$ kategorie 0 je -0.0208 a $woe_b * \beta_i$ kategorie 1 je 0.6739. Vliv této proměnné je tedy vysoký, pokud je kategorie 1, ale téměř žádný, pokud ne. Jinými slovy to, že tato proměnná predikci téměř nezměnila je vlastně poměrně podstatný vliv. Nelze tedy jednoznačně říci, že by toto vysvětlení nebylo správné.

7.2.5 SHAP

Vysvětlení metodou SHAP na obrázku 7.7 se tentokráte od LIME poměrně dost liší. SHAP totiž téměř přesně kopíruje hodnoty kterými jednotlivé proměnné k predikci ve skutečnosti přispívají, viz tabulka 7.2. Nejvýraznější rozdíl mezi LIME a SHAP pozorujeme u proměnné „x6“, kterou LIME hodnotí jako mnohem významnější. Na první pohled by se tedy dalo usoudit, že SHAP vysvětluje predikci právě pro tuto jednu zkoumanou instanci lépe.



Obrázek 7.7: SHAP *force* graf pro zkoumanou instanci.

Přesto si ale myslím, že nemůžeme spravedlivě prohlásit, že toto vysvětlení je nutně „lepší“, než vysvětlení poskytnuté metodou LIME. SHAP hodnoty

sice lépe odpoví na otázku jak přesně tato hodnota proměnné přispěla k této jedné predikci, ale LIME na druhou stranu, alespoň dle mého názoru, poskytne lepší vhled do vnitřností modelu, respektive na lokální okolí zkoumané instance. SHAP vlastně říká, že „tato hodnota proměnné přispívá k predikci přesně takto“. Naproti tomu LIME se dá vnímat trochu jako vysvětlení protipříkladem. V podstatě říká, že „změna hodnoty této proměnné by měla takový vliv“. Dle mého názoru mají obě tyto vysvětlení své výhody, ale je důležité interpretovat je správně.

Pokusím se ilustrovat na příkladu. Představme si, že proměnná „x6“ značí balíček mobilních dat – kategorie 0 znamená, že data v mobilu zákazník má, 1, že ne. Jednoho dne jsem žádal o půjčku a na základě tohoto modelu mi sice poskytnuta byla, nicméně bankovní společnost nabízí vysvětlení i v případech schválené půjčky. A tak, protože jsem velmi zvědavý zákazník, jsem si jej stejně vyžádal. Na tuto žádost mi banka poskytla výstup obou metod, LIME i SHAP.

Na první pohled jsem ohledně „x6“ velmi zmatený a ptám se, je tato proměnná důležitá nebo ne? Proto se zeptám nejprve experta na SHAP, jak tomu mám rozumět? „Toto vysvětlení znamená, že fakt, že máte balíček dat Vám skóre téměř vůbec nezlepšilo.“ Načež se tedy odeberu za expertem na LIME a zeptám se: „SHAP mi tvrdí, že mít balíček dat mi skóre téměř nezlepší, ale Vaše metoda tento fakt označuje za druhý nejdůležitější. Mýlí se tedy SHAP?“ „Ne nutně. SHAP má sice pravdu, že jenom to, že máte balíček dat Vám opravdu skóre příliš nezlepší. Nicméně pokud byste ho neměl, poměrně zásadně by Vám to uškodilo.“

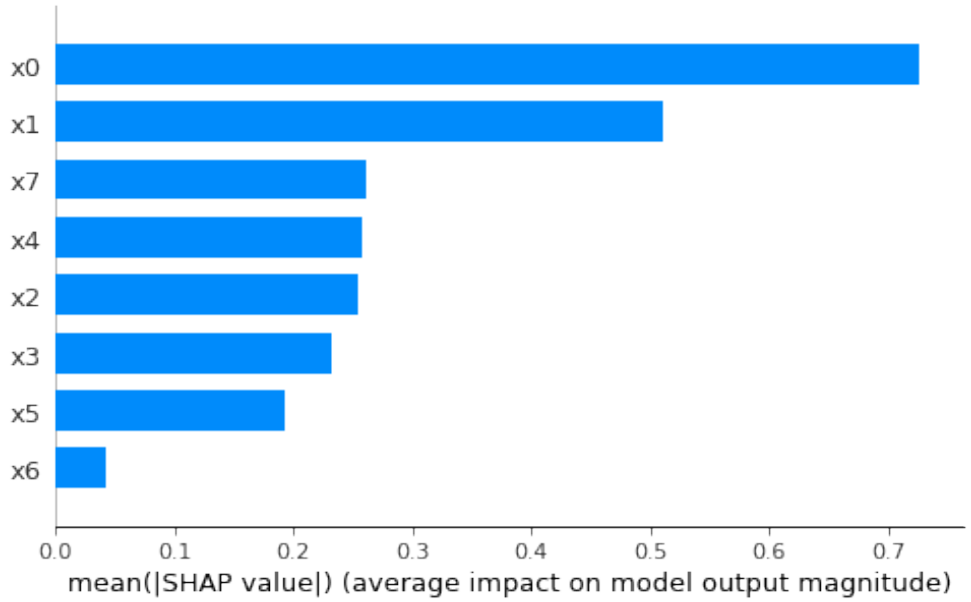
Obě vysvětlení jsou pro mě tedy užitečná, přičemž každé trochu jiným způsobem.

7.3 Spojitá data

Ukážu také, jak budou vypadat výsledky metod, pokud budeme operovat se spojitými daty. Tedy když budeme předstírat, že nevíme, že model data diskretizuje.

7.3.1 Feature importance

Na rozdíl od případu s diskretizovanými proměnnými neodpovídá feature importance založená na SHAP hodnotách tak přesně našemu očekávání. Nejmarkantnější rozdíl můžeme pozorovat u proměnné „x4“, jejíž vliv tentokrát metoda mírně podceňuje, zatímco ostatním proměnným s podobnou SHAP hodnotou trochu přidává.



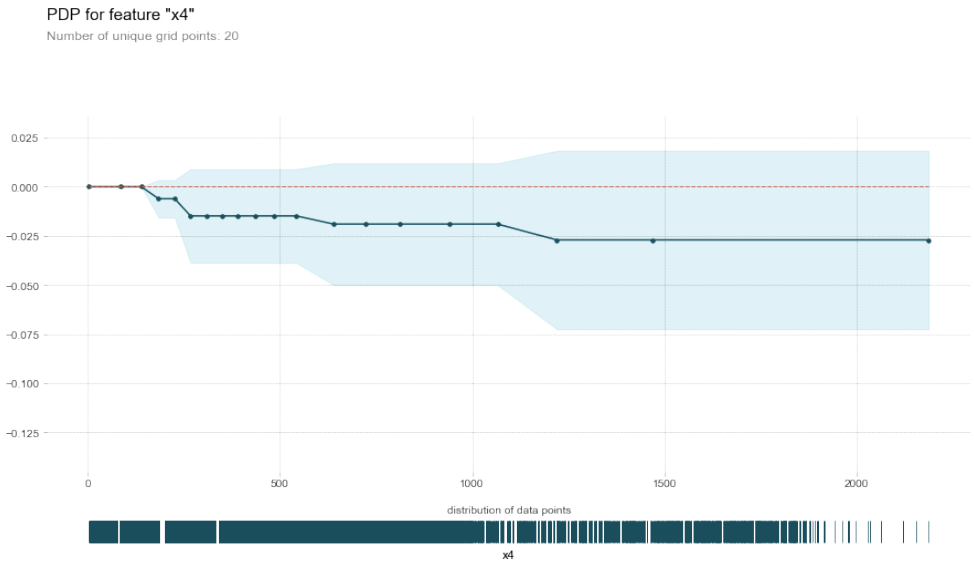
Obrázek 7.8: SHAP feature importance pro případ se spojitými hodnotami proměnných.

7.3.2 Partial dependence grafy

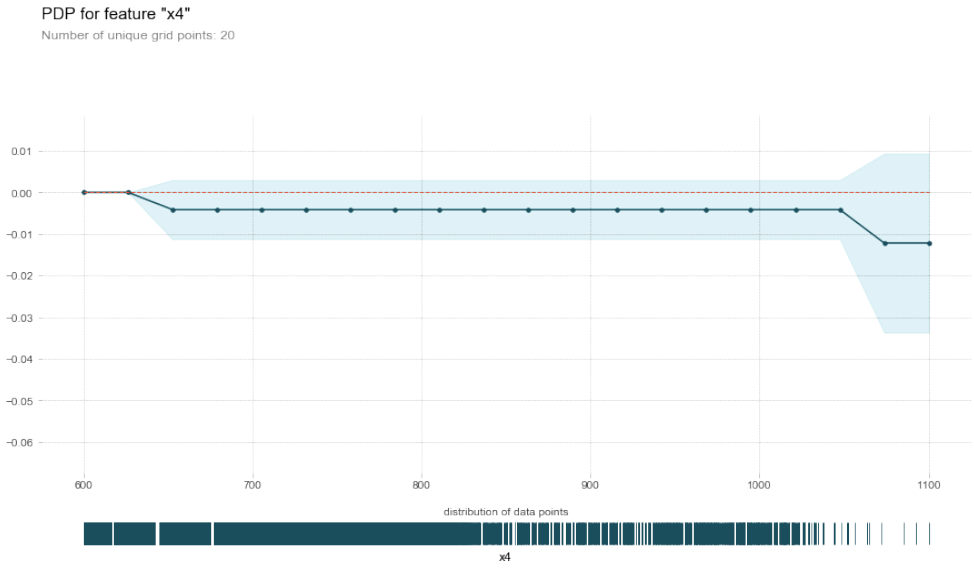
Partial dependence grafy jsou pro případ, kdy zkoumáme model, který proměnné diskretizuje a my o tom nevíme, velmi užitečné. S jejich pomocí jsme totiž schopni tuto skutečnost poměrně snadno odhalit a s trochou snahy dokonce nalézt i přesné hranice intervalů.

Na obrázku 7.9 je pd graf pro proměnnou „x4“, s horní hranicí nastavenou na 95. percentil, kvůli odstínění vlivu outlierů a vyšším počtem bodů. Už při zběžném pohledu na něj vidíme, že predikce se mění v určitých „schodech“, což téměř jistě znamená, že proměnná je diskretizovaná do binů.

Pokud se zaměříme blíže na jeden konkrétní interval, pro který je predikce konstantní a omezíme si graf na přibližně tento interval, jako na obrázku 7.10, získáme lepší představu o hranicích binu. Tuto představu můžeme zlepšovat buď dalším omezováním intervalu PD grafu nebo třeba zkoumáním konkrétní instance, které měníme hodnotu proměnné „x4“.



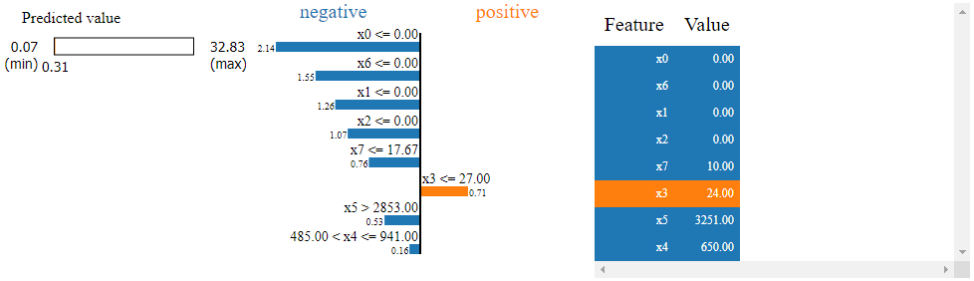
Obrázek 7.9: SHAP feature importance pro spojité proměnných.



Obrázek 7.10: Partial dependence graf proměnné „x4“ na intervalu $[600, 1100]$.

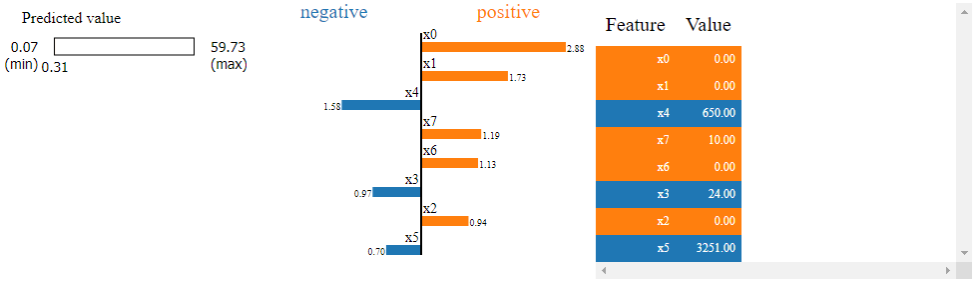
7.3.3 LIME

Metoda LIME sama spojité proměnné tak jako tak většinou diskretizuje, takže její výstup se od předchozího případu moc neliší. Jenom si můžeme povšimnout, že hranice intervalů neodpovídají přesně reálným intervalům, nicméně v tomto případě nemá tato chyba téměř žádný vliv.



Obrázek 7.11: LIME pro zkoumanou instanci se spojitými hodnotami proměnných.

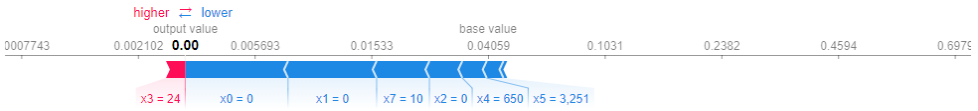
Zajímavostí je, že pokud nenecháme metodu proměnné diskretizovat, bude vysvětlení naprosto nesprávné, viz obrázek 7.12, protože se metodě nepodaří správně vytvořit vzorek dat v okolí instance tak, aby se dal natrénovat model, který by odpovídal původnímu modelu.



Obrázek 7.12: LIME pro zkoumanou instanci se spojitými hodnotami proměnných, bez jejich diskretizace.

7.3.4 SHAP

Na obrázku 7.13 vidíme, že výstup SHAP se také téměř neliší, podobně jako LIME s diskretizací si i SHAP bez problému poradil se spojitými daty.



Obrázek 7.13: SHAP pro zkoumanou instanci se spojitými hodnotami proměnných, bez jejich diskretizace.

Závěr

Cílem této práce bylo představit a porovnat metody a nástroje, či knihovny pro interpretabilitu strojového učení a na základě tohoto srovnání poté navrhnout a implementovat softwarový nástroj, který by pomohl zefektivnit validaci modelů strojového učení jak vůči businessové logice, tak požadavkům na férové strojové učení.

Nejprve jsem tedy představil samotnou problematiku interpretability modelů strojového učení. Ukázal jsem, jak ji můžeme definovat, zdůraznil její důležitost, popsal možné kategorizace metod interpretability a definoval jejich vlastnosti.

Nejpoužívanější metody a knihovny jsem následně popsal a naznačil jejich výhody, případně nevýhody. Použití vybraných metod jsem ilustroval na dvou ukázkových příkladech modelů. Prvním příkladem byl klasifikátor řešící problém predikce přežití cestujícího na lodi Titanic, přičemž druhý model řešil regresní problém odhadu ceny nemovitosti v Kalifornii. Použití dvou různých modelů mi umožnilo ukázat rozdíly při práci s interpretací regresního modelu a klasifikátoru.

Pro usnadnění práce s metodami interpretability v prostředí Jupyter notebooků jsem navrhl a implementoval knihovnu ExpyBox, která umožňuje snadné vytvoření interaktivních formulářů pro experimentování s nejpoužívanějšími metodami interpretability, mezi něž patří mimo jiné partial dependence grafy, LIME nebo SHAP. Tuto knihovnu jsem také publikoval jako open source knihovnu do repozitáře PyPI.

Použití knihovny ExpyBox jsem ilustroval na credit scoringovém modelu Vortex, respektive jeho mírně upravené kopii, z důvodu jeho citlivosti a možnosti zneužití poskytnutých informací o něm. Jelikož se jedná o logistickou regresi, mohl jsem výsledky metod, které k modelu přistupují jako k black box modelu, srovnat se skutečností. Proto jsem provedl dva experimenty, nejprve se znalostí intervalů hodnot proměnných, pro které se model chová stejně a poté bez této znalosti.

Literatura

[1] FAGGELLA, Daniel. What is Machine Learning?. In: *Emerj* [online]. Boston, February 19, 2019 [cit. 2019-06-11]. Dostupné z: https://emerj.com/ai-glossary-terms/what-is-machine-learning/

[2] FUMO, David. Types of Machine Learning Algorithms You Should Know. In: *Towards Data Science* [online]. Jun 15, 2017 [cit. 2019-06-12]. Dostupné z: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

[3] BROWNLEE, Jason. Difference Between Classification and Regression in Machine Learning. In: *Machine Learning Mastery* [online]. December 11, 2017 [cit. 2019-06-12]. Dostupné z: https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/

[4] Unsupervised Machine Learning. In: *DataRobot* [online]. [cit. 2019-06-12]. Dostupné z: https://www.datarobot.com/wiki/unsupervised-machine-learning/

[5] AlphaGo. *DeepMind* [online]. [cit. 2019-06-13]. Dostupné z: https://deepmind.com/research/alphago/

[6] OpenAI Five. *OpenAI* [online]. [cit. 2019-06-13]. Dostupné z: https://openai.com/five/

[7] DOSHI-VELEZ, Finale a Been KIM. Towards A Rigorous Science of Interpretable Machine Learning. In: *ArXiv.org* [online]. [cit. 2019-06-17]. Dostupné z: https://arxiv.org/abs/1702.08608v2

[8] OR, Biran a Cotton COURTENAY. *Explanation and Justification in Machine Learning: A Survey* [online]. [cit. 2019-06-17]. Dostupné z: https://pdfs.semanticscholar.org/02e2/e79a77d8aabc1af1900ac80ceebac20abde4.pdf

[9] KIM, Been, Rajiv KHANNA a Oluwasanmi KOYEJO. *Examples are not Enough, Learn to Criticize! Criticism for Interpretability* [online]. [cit. 2019-06-17]. Dostupné z: https://people.csail.mit.edu/beenkim/papers/KIM2016NIPS_MMD.pdf

[10] WACHTER, Sandra, Brent MITTELSTADT a Luciano FLORIDI. Why a Right to Explanation of Automated Decision-Making Does Not Exist in the General Data Protection Regulation. In: *International Data Privacy Law* [online]. 2017, **7**(2), s. 76-99 [cit. 2019-07-01]. DOI: 10.1093/idpl/ipx005. ISSN 2044-3994. Dostupné z: https://academic.oup.com/idpl/article-lookup/doi/10.1093/idpl/ipx005

[11] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)*. Dostupné také z: https://eur-lex.europa.eu/eli/reg/2016/679/oj

[12] MOLNAR, Christoph. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable* [online]. 2019 [cit. 2019-07-17]. Dostupné z: https://christophm.github.io/interpretable-ml-book/

[13] LIPTON, Zachary C. The Mythos of Model Interpretability. In: *ArXiv.org* [online]. [cit. 2019-07-09]. Dostupné z: https://arxiv.org/abs/1606.03490

[14] KAYNAR-KABUL, Ilknur. *Interpret model predictions with partial dependence and individual conditional expectation plots* [online]. [cit. 2019-07-22]. Dostupné z: https://blogs.sas.com/content/subconsciousmusings/2018/06/12/interpret-model-predictions-with-partial-dependence-and-individual-conditional-expectation-plots/

[15] FRIEDMAN, Jerome H. *Greedy function approximation: A gradient boosting machine* [online]. [cit. 2019-07-22]. Dostupné z: https://statweb.stanford.edu/~jhf/ftp/trebst.pdf

[16] GOLDSTEIN, Alex, Adam KAPELNER, Justin BLEICH a Emil PITKIN. Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation. In: *ArXiv.org* [online]. March 21, 2014 [cit. 2019-07-22]. Dostupné z: https://arxiv.org/abs/1309.6392

[17] APLEY, Daniel W. Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. In: *ArXiv.org* [online]. 27 Dec 2016 [cit. 2019-08-13]. Dostupné z: https://arxiv.org/abs/1612.08468

[18] Kolmogorov distance. *ScienceDirect* [online]. [cit. 2019-08-13]. Dostupné z: https://www.sciencedirect.com/topics/mathematics/kolmogorov-distance

[19] BREIMAN, Leo. Random Forests. In: *Machine Learning* [online]. 2001, 45:5 [cit. 2019-09-02]. DOI: 10.1023/A:1010933404324. ISSN 08856125. Dostupné z: http://link.springer.com/10.1023/A:1010933404324

[20] FISHER, Aaron, Cynthia RUDIN a Francesca DOMINICI. All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. In: *ArXiv.org* [online]. [cit. 2019-09-02]. Dostupné z: https://arxiv.org/abs/1801.01489

[21] BLEIK, Said. Permutation Feature Importance. In: *Machine Learning Blog* [online]. April 14, 2015 [cit. 2019-09-02]. Dostupné z: https://blogs.technet.microsoft.com/machinelearning/2015/04/14/permutation-feature-importance/

[22] STRINGER, Sven. Feature importance - what-s in a name?. In: *Medium* [online]. Jul 23, 2018 [cit. 2019-09-02]. Dostupné z: https://medium.com/bigdatarepublic/feature-importance-whats-in-a-name-79532e59eea3

[23] RIBEIRO, Marco Tulio, Sameer SINGH a Carlos GUESTRIN. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In: *ArXiv.org* [online]. 9 Aug 2016 [cit. 2019-09-17]. Dostupné z: https://arxiv.org/abs/1602.04938

[24] RIBEIRO, Marco Tulio, Sameer SINGH a Carlos GUESTRIN. *Anchors: High-Precision Model-Agnostic Explanations* [online]. 2018 [cit. 2019-10-08]. Dostupné z: https://homes.cs.washington.edu/~marcotcr/aaai18.pdf

[25] KAUFMANN, Emilie a Shivaram KALYANAKRISHNAN. Information Complexity in Bandit Subset Selection. In: *JMLR: Workshop and Conference Proceedings vol 30* [online]. 2013 [cit. 2019-10-10]. Dostupné z: http://proceedings.mlr.press/v30/Kaufmann13.pdf

[26] SHAPLEY, L. S., Harold William KUHN a Albert William TUCKER. 17. A Value for n-Person Games. In: *Contributions to the Theory of Games (AM-28), Volume II* [online]. Princeton: Princeton University Press, 1953, 1953-12-31, s. 307-318 [cit. 2019-10-18]. DOI: 10.1515/9781400881970-018. ISBN 9781400881970. Dostupné z: http://www.degruyter.com/view/books/9781400881970/9781400881970-018/9781400881970-018.xml

[27] ŠTRUMBELJ, Erik a Igor KONONENKO. Explaining prediction models and individual predictions with feature contributions. *Knowledge*

and Information Systems [online]. 2014, **41**(3), 647-665 [cit. 2019-10-21]. DOI: 10.1007/s10115-013-0679-x. ISSN 0219-1377. Dostupné z: http://link.springer.com/10.1007/s10115-013-0679-x

[28] LUNDBERG, Scott M. a Su-In LEE. A Unified Approach to Interpreting Model Predictions. In: *Advances in Neural Information Processing Systems 30 (NIPS 2017)* [online]. [cit. 2019-10-24]. Dostupné z: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictionsRibeiro

[29] LUNDBERG, Scott M., Gabriel G. ERION a Su-In LEE. Consistent Individualized Feature Attribution for Tree Ensembles. In: *ArXiv.org* [online]. 12 Feb 2018 [cit. 2019-10-24]. Dostupné z: https://arxiv.org/abs/1802.03888

[30] WACHTER, Sandra, Brent MITTELSTADT a Chris RUSSELL. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. In: *ArXiv.org* [online]. 1 Nov 2017 [cit. 2019-11-11]. Dostupné z: https://arxiv.org/abs/1711.00399

[31] DHURANDHAR, Amit, Pin-Yu CHEN, Ronny LUSS, Chun-Chen TU, Paishun TING, Karthikeyan SHANMUGAM a Payel DAS. Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives. In: *ArXiv.org* [online]. 21 Feb 2018 [cit. 2019-11-15]. Dostupné z: https://arxiv.org/abs/1802.07623

[32] LOOVEREN, Arnaud Van a Janis KLAISE. Interpretable Counterfactual Explanations Guided by Prototypes. In: *ArXiv.org* [online]. 3 Jul 2019 [cit. 2019-11-15]. Dostupné z: https://arxiv.org/abs/1907.02584

[33] awesome-machine-learning-interpretability. *GitHub* [online]. [cit. 2020-01-06]. Dostupné z: https://github.com/jphall663/awesome-machine-learning-interpretability

[34] MOLNAR, Christoph. Iml: interpretable machine learning. *GitHub* [online]. [cit. 2019-12-03]. Dostupné z: https://github.com/christophM/iml

[35] Descriptive mAchine Learning EXplanations. *GitHub* [online]. [cit. 2019-12-03]. Dostupné z: https://github.com/ModelOriented/DALEX

[36] STANIAK, Mateusz a Przemyslaw BIECEK. Explanations of model predictions with live and breakDown packages. In: *ArXiv.org* [online]. [cit. 2019-12-03]. Dostupné z: https://arxiv.org/abs/1804.01955

[37] Skater. *GitHub* [online]. [cit. 2019-12-03]. Dostupné z: https://github.com/oracle/Skater

[38] Alibi. *GitHub* [online]. [cit. 2019-12-03]. Dostupné z: https://github.com/SeldonIO/alibi

[39] AI Explainability 360. *GitHub* [online]. [cit. 2019-12-03]. Dostupné z: https://github.com/IBM/AIX360

[40] HIND, Michael, Dennis WEI, Murray CAMPBELL, Noel C. F. CODELLA, Amit DHURANDHAR, Aleksandra MOJSILOVIĆ, Karthikeyan Natesan RAMAMURTHY a Kush R. VARSHNEY. TED: Teaching AI to Explain its Decisions. In: *ArXiv.org* [online]. [cit. 2019-12-04]. Dostupné z: https://arxiv.org/abs/1811.04896

[41] AI Explainability 360. *IBM Research Trusted AI* [online]. [cit. 2019-12-04]. Dostupné z: https://aix360.mybluemix.net/

[42] AI Fairness 360. *IBM Research Trusted AI* [online]. [cit. 2019-12-04]. Dostupné z: http://aif360.mybluemix.net/

[43] Lime. *GitHub* [online]. [cit. 2019-12-05]. Dostupné z: https://github.com/marcotcr/lime

[44] Shap. *GitHub* [online]. [cit. 2019-12-05]. Dostupné z: https://github.com/slundberg/shap

[45] PDPbox. *GitHub* [online]. [cit. 2019-12-05]. Dostupné z: https://github.com/SauceCat/PDPbox/

[46] Titanic: Machine Learning from Disaster. *Kaggle* [online]. [cit. 2019-12-09]. Dostupné z: https://www.kaggle.com/c/titanic

[47] Eli5. *Github* [online]. [cit. 2019-12-10]. Dostupné z: https://github.com/TeamHG-Memex/eli5

[48] *California Housing* [online]. [cit. 2019-12-27]. Dostupné z: https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

[49] Jupyter kernels. *GitHub* [online]. [cit. 2019-12-31]. Dostupné z: https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

[50] ipywidgets: Interactive HTML Widgets. *GitHub* [online]. [cit. 2020-01-05]. Dostupné z: https://github.com/jupyter-widgets/ipywidgets

[51] *Read the Docs* [online]. [cit. 2020-01-05]. Dostupné z: https://readthedocs.org/

[52] PyPI. *The Python Package Index* [online]. [cit. 2020-01-06]. Dostupné z: https://pypi.org/

Příloha A

Obsah přiloženého CD

Zdrojová forma práce ve formátu LaTeX a přiložené Jupyter notebooky jsou k dispozici také na `https://gitlab.fit.cvut.cz/stercjak/dp`.

Zdrojové kódy knihovny ExpyBox jsou zveřejněny na adrese `https://github.com/Kukuksumusu/expybox`.

- `DP`
  - `src` ..... zdrojová forma práce ve formátu LaTeX
  - `DP_Stercl_Jakub_2020.pdf` ..... text práce ve formátu PDF
  - `JupyterNTBS` ..... adresář se zdrojovými Jupyter notebooky
    - `CaliforniaHousing` ..... California Housing dataset
    - `titanic_data` ..... Titanic dataset
    - `Vortex` ..... Vortex data a implementace
      - `vortex_anonymized.py` ..... implementace modelu Vortex
    - `Housing.ipynb` ..... notebook pro dataset California Housing
    - `Titanic.ipynb` ..... notebook pro dataset Titanic
    - `Vortex-binned.ipynb` .... notebook pro Vortex s diskretizovanými hodnotami proměnných
    - `Vortex-unbinned.ipynb` ..... notebook pro Vortex se spojitými hodnotami proměnných
- `expybox` ..... knihovna ExpyBox
  - `docs` ..... zdrojové kódy dokumentace
  - `expybox` ..... zdrojové kódy knihovny ExpyBox