BI-VZD přednáška 2

Ondřej Tichý

FIT ČVUT

27.2.2023

Autoři: Karel Klouda, Juan Pablo Maldonado Lopez, Daniel Vašata.
Problémy, návrhy apod. hlaste v GitLabu.
Verze souboru: 27. února 2023 13:35.

Co bude v této přednášce

- hyperparametry a testovací, trénovací a validační data
- jak na spojité příznaky
- jak na spojitou vysvětlovanou proměnnou

Jak je můj strom dobrý?

Nyní se dostáváme k druhé otázce: **Jak poznám, že můj vytvořený strom není jen dobrým modelem dat, která mám, ale bude dobře fungovat i pro data jiná?**

Jak je můj strom dobrý?

Nyní se dostáváme k druhé otázce: **Jak poznám, že můj vytvořený strom není jen dobrým modelem dat, která mám, ale bude dobře fungovat i pro data jiná?**

- Chceme-li rozhodovat, jak je model dobrý, potřebujeme nějakou objektivní vyčíslitelnou míru jeho kvality.
- Volba této míry je důležitou součástí celého procesu hledání modelu. Míry se obvykle velmi liší pro problémy klasifikace a regrese.

Jak je můj strom dobrý?

Nyní se dostáváme k druhé otázce: **Jak poznám, že můj vytvořený strom není jen dobrým modelem dat, která mám, ale bude dobře fungovat i pro data jiná?**

- Chceme-li rozhodovat, jak je model dobrý, potřebujeme nějakou objektivní vyčíslitelnou míru jeho kvality.
- Volba této míry je důležitou součástí celého procesu hledání modelu. Míry se obvykle velmi liší pro problémy klasifikace a regrese.
- My se nyní věnujeme klasifikaci a vystačíme si s přirozenou mírou **klasifikační přesnosti** (angl. **classification accuracy**), která prostě měří poměr správně klasifikovaných, tedy je rovna číslu (příp. procentu)

$$\frac{\text{počet správně klasifikovaných dat}}{\text{počet všech dat}}.$$

- Např. rozhodovací strom zkonstruovaný algoritmem ID3 pro množinu osmi dat se pletl ve dvou případech, a tedy jeho přesnost byla $\frac{6}{8} = 0.75 = 75\ \%$.

Jak je můj strom dobrý? Jakože opravdu?

- Mohlo by se zdát, že máme vyřešeno: prostě zkonstruujeme strom, který má pro naše data nejvyšší přesnost.

Jak je můj strom dobrý? Jakože opravdu?

- Mohlo by se zdát, že máme vyřešeno: prostě zkonstruujeme strom, který má pro naše data nejvyšší přesnost.
- S tímto přístupem bychom ale narazili, neb by vedl k hlubokým stromům, které mají na listech třeba jen jeden datový bod.
- Platí totiž, že čím je strom hlubší, tím má lepší přesnost!
- My vlastně nechceme maximalizovat přesnost na našich datech, ale **na všech možných datech**, která ovšem nemáme.

⑦ Jak to vyřešit?

Jak je můj strom dobrý? Jakože opravdu?

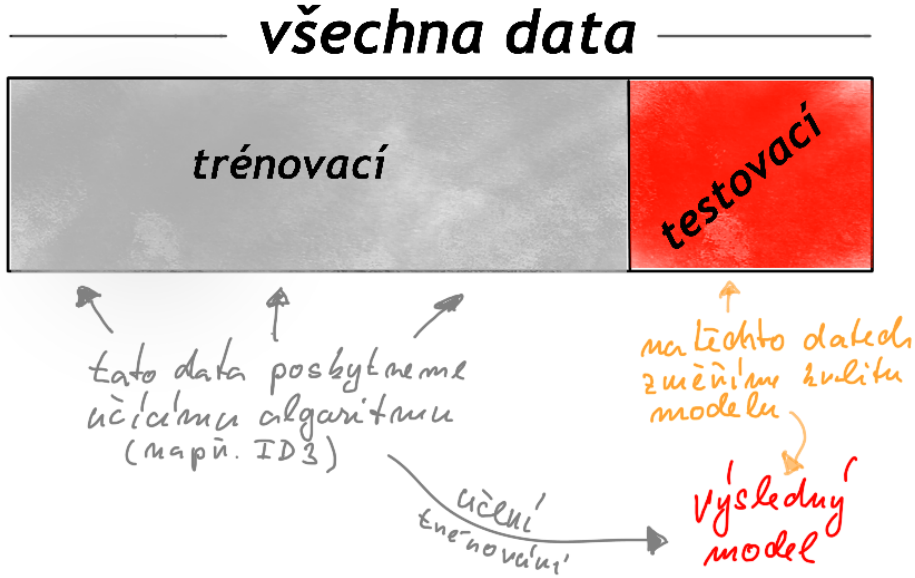
- Mohlo by se zdát, že máme vyřešeno: prostě zkonstruujeme strom, který má pro naše data nejvyšší přesnost.
- S tímto přístupem bychom ale narazili, neb by vedl k hlubokým stromům, které mají na listech třeba jen jeden datový bod.
- Platí totiž, že čím je strom hlubší, tím má lepší přesnost!
- My vlastně nechceme maximalizovat přesnost na našich datech, ale **na všech možných datech**, která ovšem nemáme.

(?) Jak to vyřešit?

- Uděláme to tak, že naše data rozdělíme na dva kusy: na jednom (tom větším), model naučíme (např. ID3 algoritmem) a na tom druhém kusu změříme přesnost. **Tak dostaneme spolehlivější odhad toho, jak se bude náš model chovat pro data, na kterých se neučil.**

Trénovací a testovací data

- Těm dvěma kusům dat se obvykle říká **trénovací** a **testovací data** (angl. **train set** a **test set**).
- Chybovost modelu (pro nás nepřesnost = 1 - přesnost) na těchto množinách dat se pak adekvátně říká **trénovací** a **testovací chyba** (angl. **train error** a **test error**).



Přeučení modelu (1/2)

- V případě stromů zjevně platí, že čím hlubší strom učíme, tím dostaneme menší trénovací chybu. Spolehlivější měrou kvality je však testovací chyba, neb tou trénovací si lžeme do kapsy: Měříme, jak dobře jsme *přizpůsobili* strom dostupným datům, nikoli jak dobře jsme odhadli skrytý model za těmito daty schovaným (pokud tedy na takový model věříte).

- Tomuto přílišnému přizpůsobení trénovacím datům se říká **přeučení** modelu (angl. **overfitting**).

- Obvykle platí, že čím složitější model (v našem příp. čím hlubší strom), tím nižší trénovací chyba.

Přeučení modelu (1/2)

- V případě stromů zjevně platí, že čím hlubší strom učíme, tím dostaneme menší trénovací chybu. Spolehlivější měrou kvality je však testovací chyba, neb tou trénovací si lžeme do kapsy: Měříme, jak dobře jsme *přizpůsobili* strom dostupným datům, nikoli jak dobře jsme odhadli skrytý model za těmito daty schovaným (pokud tedy na takový model věříte).

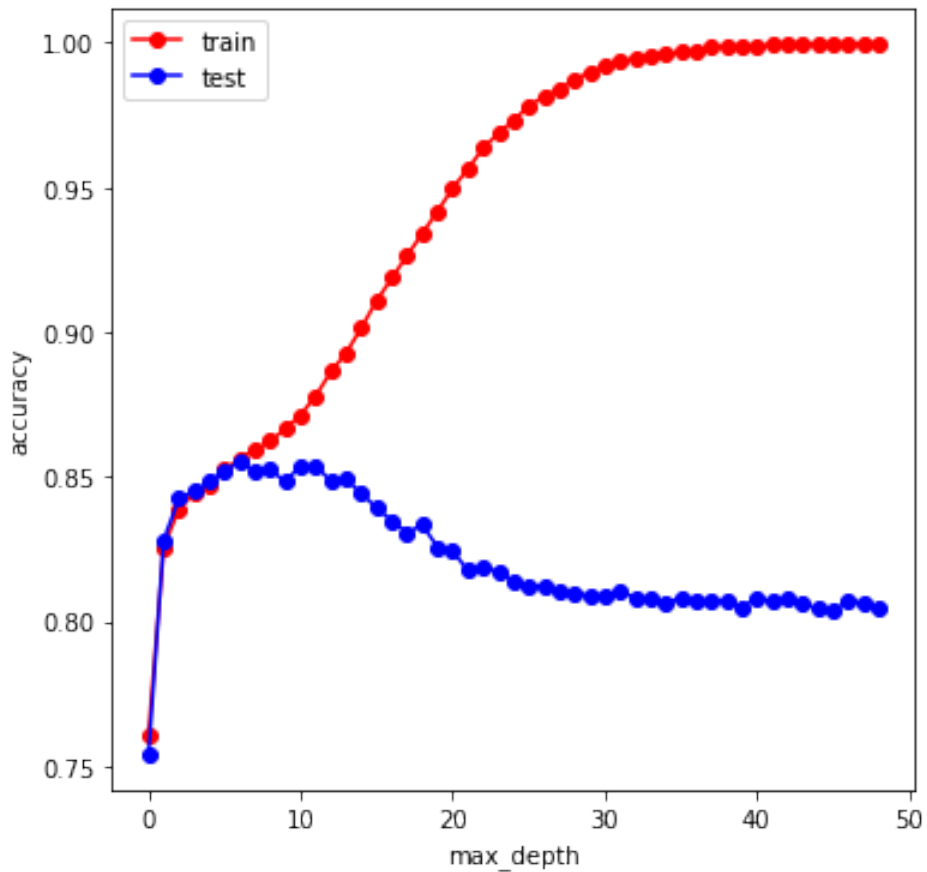
- Tomuto přílišnému přizpůsobení trénovacím datům se říká **přeučení** modelu (angl. **overfitting**).

- Obvykle platí, že čím složitější model (v našem příp. čím hlubší strom), tím nižší trénovací chyba.

- Testovací chyba se však chová jinak: Nejdříve se zvětšováním složitosti modelu klesá, ale v jistý okamžik začne růst. Najít tento bod zlomu je úkolem celého procesu budování modelu.

Přeučení modelu (2/2)

- Následující obrázek ukazuje typický vývoj trénovací a testovací chyby v závislosti na hloubce stromu (`max_depth`).
- Z tohoto obrázku je vidět, že nejrozumnější volba parametru `max_depth` je 6.



Ladění hyperparametrů (1/3)

Teď se dostáváme ke třetí otázce: **Jak poznám, jakou mám zvolit hloubku stromu příp. jiné jeho parametry?**

Ladění hyperparametrů (1/3)

Teď se dostáváme ke třetí otázce: **Jak poznám, jakou mám zvolit hloubku stromu příp. jiné jeho parametry?**

- Můžeme postupovat takto:
  1. Data si rozdělíme na trénovací a testovací.
  2. Pro různé hodnoty hloubky stromu (parametr `max_depth`) naučíme rozhodovací strom na trénovacích datech
  3. a pro každou hloubku stromu změříme přesnost (classification accuracy) na testovacích datech.
  4. Vyberme tu hloubku, která dává nejmenší testovací chybu.
  5. Tuto hodnotu vezmeme také jako odhad chyby na reálných datech, a tedy jako objektivní míru kvality modelu.
- Je někde problém?

Ladění hyperparametrů (1/3)

Teď se dostáváme ke třetí otázce: **Jak poznám, jakou mám zvolit hloubku stromu příp. jiné jeho parametry?**

- Můžeme postupovat takto:
  1. Data si rozdělíme na trénovací a testovací.
  2. Pro různé hodnoty hloubky stromu (parametr `max_depth`) naučíme rozhodovací strom na trénovacích datech
  3. a pro každou hloubku stromu změříme přesnost (classification accuracy) na testovacích datech.
  4. Vyberme tu hloubku, která dává nejmenší testovací chybu.
  5. Tuto hodnotu vezmeme také jako odhad chyby na reálných datech, a tedy jako objektivní míru kvality modelu.
- Je někde problém? Ano, je...
- Výsledná testovací chyba bude zpravidla příliš optimistickým odhadem skutečnosti! Výsledný model (konkrétně parametr hloubka stromu) byl vybrán na základě těchto dat a model je tedy těmto datům přizpůsobený.
- Takto bychom porušili zásadu, že **při učení modelu nesaháme na testovací data**, pokud má být testovací chyba rozumným odhadem skutečné chyby modelu.

Ladění hyperparametrů (2/3)

- Jak se s tímto problémem vypořádat?

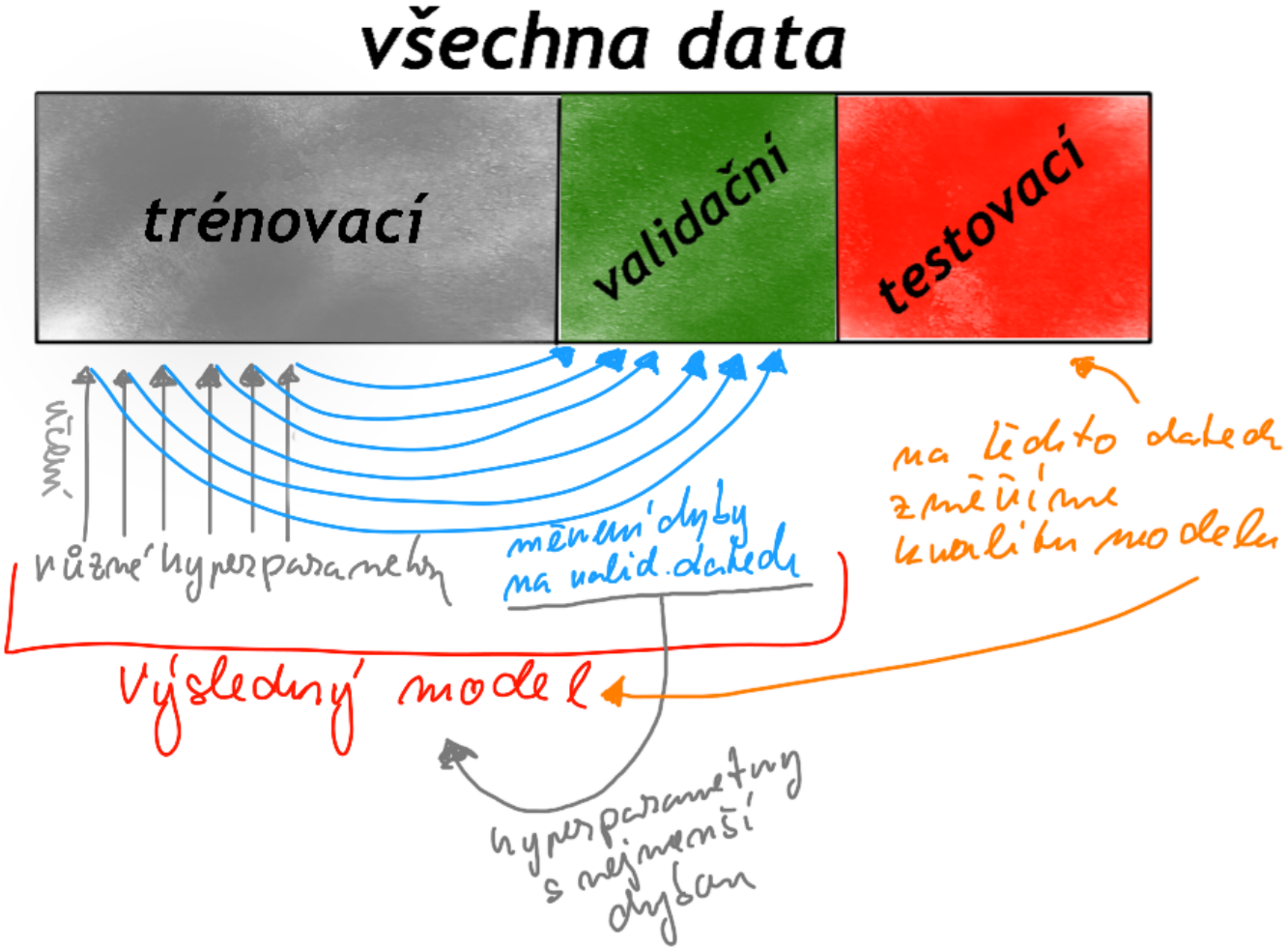
Ladění hyperparametrů (2/3)

- Jak se s tímto problémem vypořádat?
- Obvykle se to dělá tak, že se data rozdělí ne na dvě, ale na tři podmnožiny: trénovací, **validační** (angl. **validation**) a testovací:
  1. Pro různé hodnoty hloubky stromu `max_depth` naučíme rozhodovací strom
  2. a změříme jeho chybu (přesnost) na validačních datech.
  3. Jako optimální hodnotu vybereme tu s nejmenší chybou (tj. s nejvyšší přesností).
  4. Chyba na testovacích datech, které dosud ležely ladem, je pak rozumným odhadem chyby modelu.

Ladění hyperparametrů (2/3)

- Jak se s tímto problémem vypořádat?
- Obvykle se to dělá tak, že se data rozdělí ne na dvě, ale na tři podmnožiny: trénovací, **validační** (angl. **validation**) a testovací:
  1. Pro různé hodnoty hloubky stromu `max_depth` naučíme rozhodovací strom
  2. a změříme jeho chybu (přesnost) na validačních datech.
  3. Jako optimální hodnotu vybereme tu s nejmenší chybou (tj. s nejvyšší přesností).
  4. Chyba na testovacích datech, které dosud ležely ladem, je pak rozumným odhadem chyby modelu.
- Parametrům, jako je `max_depth`, které určují *tvar* nebo *komplexitu* modelu, se říká **hyperparametry modelu**.
- Určování těchto parametrů pomocí validačních dat se říká **ladění** (angl. **tuning**).
- Tento parametr nemusí být jeden, ale může jich být více. Pokud jsou spojité, může být výpočetně složité jich otestovat *reprezentativní* množství a je třeba dělat kompromisy.

Ladění hyperparametrů (3/3)



Ladění hyperparametrů: poznámky (1/2)

- Pro představu, jaké hyperparametry jsou dostupné pro rozhodovací stromy v knihovně `sklearn`:

```
Init signature: DecisionTreeClassifier(criterion='gini', splitter='best', max_de
pth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
 max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decreas
e=0.0, min_impurity_split=None, class_weight=None, presort=False)
```

Ladění hyperparametrů: poznámky (1/2)

- Pro představu, jaké hyperparametry jsou dostupné pro rozhodovací stromy v knihovně `sklearn`:

```
Init signature: DecisionTreeClassifier(criterion='gini', splitter='best', max_de
pth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
 max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decreas
e=0.0, min_impurity_split=None, class_weight=None, presort=False)
```

- Význam (některých) těchto parametrů si vysvětlíte (a ukážeme) na cvičení.

Ladění hyperparametrů: poznámky (1/2)

- Pro představu, jaké hyperparametry jsou dostupné pro rozhodovací stromy v knihovně `sklearn`:

```
Init signature: DecisionTreeClassifier(criterion='gini', splitter='best', max_de
pth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
 max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decreas
e=0.0, min_impurity_split=None, class_weight=None, presort=False)
```

- Význam (některých) těchto parametrů si vysvětlíte (a ukážeme) na cvičení.
- Dělení dat na trénovací/validační/testovací podmnožiny je dobré dělat náhodně. Tj. nevybrat první půlku dat jako trénovací, další čtvrtinu jako validační a zbytek vzít jako testovací.
- V pořadí dat by mohla být nějaká zákonitost, která by znamenala, že trénovací a testovací data nebudou reprezentativní.

Ladění hyperparametrů: poznámky (1/2)

- Pro představu, jaké hyperparametry jsou dostupné pro rozhodovací stromy v knihovně `sklearn`:

```
Init signature: DecisionTreeClassifier(criterion='gini', splitter='best', max_de
pth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
 max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decreas
e=0.0, min_impurity_split=None, class_weight=None, presort=False)
```

- Význam (některých) těchto parametrů si vysvětlíte (a ukážeme) na cvičení.
- Dělení dat na trénovací/validační/testovací podmnožiny je dobré dělat náhodně. Tj. nevybrat první půlku dat jako trénovací, další čtvrtinu jako validační a zbytek vzít jako testovací.
- V pořadí dat by mohla být nějaká zákonitost, která by znamenala, že trénovací a testovací data nebudou reprezentativní.
- Proto se postupuje tak, že se data vybírají náhodně. V `sklearn` je na to metoda:

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.25, random_state=33)
```

Ladění hyperparametrů: poznámky (2/2)

- Neexistuje žádný optimální poměr velikosti trénovací/validační/testovací množiny dat.
- Obvyklé poměry jsou takové, že 20 % dat vezmeme jako testovací množinu a ze zbytku vezmeme 20 % jako validační množinu. Co zbude, jsou trénovací data.
- Místo 20 % lze použít 25 %, nebo 30 %.
- Lze použít i sofistikovanější strategie, např. měnit velikost validační množiny a koukat se, co to dělá.

Ladění hyperparametrů: poznámky (2/2)

- Neexistuje žádný optimální poměr velikosti trénovací/validační/testovací množiny dat.
- Obvyklé poměry jsou takové, že 20 % dat vezmeme jako testovací množinu a ze zbytku vezmeme 20 % jako validační množinu. Co zbude, jsou trénovací data.
- Místo 20 % lze použít 25 %, nebo 30 %.
- Lze použít i sofistikovanější strategie, např. měnit velikost validační množiny a koukat se, co to dělá.
- Často používanou metodou je také **cross-validace**, o které budeme mluvit později.

Vícehodnotové příznaky: one-hot encoding

- Mnohé implementace rozhodovacích stromů (vč. té v `sklearn`) umí konstruovat pouze binární stromy.
- Neumí si tak řádně poradit s diskrétními (**kategorickými**) příznaky, které mají více než dvě různé hodnoty.

Vícehodnotové příznaky: one-hot encoding

- Mnohé implementace rozhodovacích stromů (vč. té v `sklearn`) umí konstruovat pouze binární stromy.
- Neumí si tak řádně poradit s diskrétními (**kategorickými**) příznaky, které mají více než dvě různé hodnoty.
- Tento problém lze vyřešit pomocí **one-hot encoding**, kdy se jeden kategoriální příznak s n různými hodnotami nahradí n binárními **dummy variables**.
- Např. v našem příkladu s rýmičkou bychom mohli mít tři možnosti: vstal(a), nevstal(a), spadl(a) z postele. Abychom toto převedli do binárních příznaků pomoci dummy příznaků, museli bychom zavést tři nové binární příznaky *vstal(a) / nevstal(a) / spadl(a)*:

původní příznak	$\rightarrow$ *dummy příznaky* $\rightarrow$	vstal	nevstal	spadl
vstal	$\longrightarrow$	1	0	0
nevstal	$\longrightarrow$	0	1	0
spadl	$\longrightarrow$	0	0	1

Nominální a ordinální příznaky

- Metoda *one-hot encoding* má své nevýhody:

Nominální a ordinální příznaky

- Metoda *one-hot encoding* má své nevýhody:
  - Zvyšuje výrazně počet příznaků (dimenzi datasetu), což speciálně u rozhodovacích stromů může výrazně podporovat přeučení.
  - Z datasetu sice nemizí žádná informace, ale vzhledem k tomu, že algoritmus pro konstrukci stromů není optimální (jen „hladový“), může se zvýšení počtu příznaků projevit na kvalitě modelu.
  - Není vhodná pro všechny druhy kategoriálních příznaků, zejm. ne pro ty, kde je mezi jednotlivými kategoriemi nějaké pořadí.

Nominální a ordinální příznaky

- Metoda *one-hot encoding* má své nevýhody:
  - Zvyšuje výrazně počet příznaků (dimenzi datasetu), což speciálně u rozhodovacích stromů může výrazně podporovat přeučení.
  - Z datasetu sice nemizí žádná informace, ale vzhledem k tomu, že algoritmus pro konstrukci stromů není optimální (jen „hladový“), může se zvýšení počtu příznaků projevit na kvalitě modelu.
  - Není vhodná pro všechny druhy kategoriálních příznaků, zejm. ne pro ty, kde je mezi jednotlivými kategoriemi nějaké pořadí.

- Kategoriální příznaky jsou v zásadě dvou druhů:

  Nominální Mezi kategoriemi není žádné pořadí, např. *místo narození*, *fakulta*, *pohlaví*, atp.

  Ordinální Mezi kategoriemi je nějaké přirozené uspořádání, např. *vzdělání (základní* $<$ *střední* $<$ *s maturitou* $<$ *univerzitní)* atp.

Nominální a ordinální příznaky

- Metoda *one-hot encoding* má své nevýhody:
  - Zvyšuje výrazně počet příznaků (dimenzi datasetu), což speciálně u rozhodovacích stromů může výrazně podporovat přeučení.
  - Z datasetu sice nemizí žádná informace, ale vzhledem k tomu, že algoritmus pro konstrukci stromů není optimální (jen „hladový“), může se zvýšení počtu příznaků projevit na kvalitě modelu.
  - Není vhodná pro všechny druhy kategoriálních příznaků, zejm. ne pro ty, kde je mezi jednotlivými kategoriemi nějaké pořadí.

- Kategoriální příznaky jsou v zásadě dvou druhů:

  Nominální Mezi kategoriemi není žádné pořadí, např. *místo narození*, *fakulta*, *pohlaví*, atp.

  Ordinální Mezi kategoriemi je nějaké přirozené uspořádání, např. *vzdělání (základní* $<$ *střední* $<$ *s maturitou* $<$ *univerzitní)* atp.

- Metodu *one-hot encoding* je tedy lepší používat pouze pro nominální příznaky. Ordinální je lepší nechat tak jak jsou, `sklearn` s nimi může pracovat jako se spojitými příznaky, což dává větší smysl (viz dále).

Spojité příznaky

- V datech ale většinou nemáme pouze kategoriální příznaky. Často se v nich objevují příznaky jako *věk*, *cena*, *teplota*, *tlak* atp., které je lepší chápat jako příznaky **spojité** (angl. **continuous**).

Spojité příznaky

- V datech ale většinou nemáme pouze kategoriální příznaky. Často se v nich objevují příznaky jako *věk*, *cena*, *teplota*, *tlak* atp., které je lepší chápat jako příznaky **spojité** (angl. **continuous**).

- V rozhodovacích stromech mohou k větvení sloužit i tyto příznaky. Pravidlo pro (binární) větvení může být např. `věk` < 30, které opět množinu dat rozdělí na dvě části.

- Oproti binárním příznakům má smysl, aby se spojité proměnné objevily ve stromu vícekrát. Rozdělíme-li data pravidlem `věk` < 30, dává smysl „starší“ část dat dělit znovu pravidlem `věk` < 60. U pravidla s binárním příznakem `pohlaví == žena` toto smysl nedávalo.

Spojité příznaky a algoritmus

- Algoritmus pro binární příznak $X \in \{0, 1\}$ fungoval tak, že spočítal *informační zisk* rozdělení dat $\mathcal{D}$ podle pravidla $X == 0$ a buď toto dělení použil, nebo si vybral jiný příznak s vyšším informačním ziskem.

- Spojitý příznak X s hodnotami např. z intervalu $[0, 100]$ musí fungovat jinak, neboť pravidel tvaru $X < d$ existuje vlastně nekonečno (pro všechna možná $0 < d \leq 100$).

Spojité příznaky a algoritmus

- Algoritmus pro binární příznak $X \in \{0, 1\}$ fungoval tak, že spočítal *informační zisk* rozdělení dat $\mathcal{D}$ podle pravidla $X == 0$ a buď toto dělení použil, nebo si vybral jiný příznak s vyšším informačním ziskem.

- Spojitý příznak X s hodnotami např. z intervalu $[0, 100]$ musí fungovat jinak, neboť pravidel tvaru $X < d$ existuje vlastně nekonečno (pro všechna možná $0 < d \leq 100$).

- V základním nastavení algoritmus funguje tak, že ale vlastně všechna dělení ve tvaru $X < d$ vyzkouší a vybere to s nejvyšším informačním ziskem. Funguje to takto:
  1. Projdi všechny možné hodnoty příznaku X v právě dělené množině dat $\mathcal{D}$ a setřiď je podle velikosti: $x_0 < x_1 < \cdots < x_\ell$.
  2. Vyzkoušej rozdělení dat podle pravidla $X < x_i$ pro všechna $i = 1, 2, \ldots, \ell$ a pro každé takové rozdělení $\mathcal{D}$ spočítej informační zisk.
  3. Jako nejlepší pravidlo dělení podle příznaku X vezmi to s největším spočítaným informačním ziskem.

Spojité příznaky a algoritmus: příklad

Představme si, že máme data $\mathcal{D}$ s osmi řádky s příznakem `věk`, který má pro řádky 0 až 7 tyto hodnoty:

$$12_0, 35_1, 15_2, 65_3, 35_4, 15_5, 20_6, 20_7,$$

přičemž binární vysvětlovaná proměnná Y má pro tyto řádky hodnoty

$$0_0, 1_1, 0_2, 0_3, 1_4, 0_5, 1_6, 1_7.$$

Entropie $\mathcal{D}$ je tedy maximální, tj. $H(\mathcal{D}) = 1$.

Spojité příznaky a algoritmus: příklad

Představme si, že máme data $\mathcal{D}$ s osmi řádky s příznakem `věk`, který má pro řádky 0 až 7 tyto hodnoty:

$$12_0, 35_1, 15_2, 65_3, 35_4, 15_5, 20_6, 20_7,$$

přičemž binární vysvětlovaná proměnná Y má pro tyto řádky hodnoty

$$0_0, 1_1, 0_2, 0_3, 1_4, 0_5, 1_6, 1_7.$$

Entropie $\mathcal{D}$ je tedy maximální, tj. $H(\mathcal{D}) = 1$.
Dle předchozího postupujeme tak, že možné hodnoty X seřadíme:
$12, 15, 20, 35, 65$, a vyzkoušíme všechna následující rozdělení.

pravidlo	$\mathcal{D}_L$	$\mathcal{D}_R$	informační zisk
$X < 15$	0_0	$1_1, 0_2, 0_3, 1_4, 0_5, 1_6, 1_7$	0.14
$X < 20$	$0_0, 0_2, 0_5$	$1_1, 0_3, 1_4, 1_6, 1_7$	0.55
$X < 35$	$0_0, 0_2, 0_5, 1_6, 1_7$	$1_1, 0_3, 1_4$	0.05
$X < 65$	$0_0, 1_1, 0_2, 1_4, 0_5, 1_6, 1_7$	0_3	0.14

Spojité příznaky a algoritmus: příklad

Představme si, že máme data $\mathcal{D}$ s osmi řádky s příznakem `věk`, který má pro řádky 0 až 7 tyto hodnoty:

$$12_0, 35_1, 15_2, 65_3, 35_4, 15_5, 20_6, 20_7,$$

přičemž binární vysvětlovaná proměnná Y má pro tyto řádky hodnoty

$$0_0, 1_1, 0_2, 0_3, 1_4, 0_5, 1_6, 1_7.$$

Entropie $\mathcal{D}$ je tedy maximální, tj. $H(\mathcal{D}) = 1$.
Dle předchozího postupujeme tak, že možné hodnoty X seřadíme:
$12, 15, 20, 35, 65$, a vyzkoušíme všechna následující rozdělení.

pravidlo	$\mathcal{D}_L$	$\mathcal{D}_R$	informační zisk
$X < 15$	0_0	$1_1, 0_2, 0_3, 1_4, 0_5, 1_6, 1_7$	0.14
$X < 20$	$0_0, 0_2, 0_5$	$1_1, 0_3, 1_4, 1_6, 1_7$	0.55
$X < 35$	$0_0, 0_2, 0_5, 1_6, 1_7$	$1_1, 0_3, 1_4$	0.05
$X < 65$	$0_0, 1_1, 0_2, 1_4, 0_5, 1_6, 1_7$	0_3	0.14

Pokud se tedy nenajde jiný příznak dávající lepší informační zisk, použije algoritmus pro rozdělení dat $\mathcal{D}$ pravidlo $X < 20$.

Poznámky

- Pokud má spojitý příznak příliš mnoho hodnot, obvykle se postupuje tak, že se uvažuje pravidlo $X < x_i$ například pouze pro každé desáté x_i. Příp. se náhodně vyzkouší daný počet hodnot.
- Implementace v `sklearn` se chová vlastně ke všem příznakům jako ke spojitým.
- Dovede si tak celkem dobře poradit s ordinálními příznaky.

Poznámky

- Pokud má spojitý příznak příliš mnoho hodnot, obvykle se postupuje tak, že se uvažuje pravidlo $X < x_i$ například pouze pro každé desáté x_i. Příp. se náhodně vyzkouší daný počet hodnot.
- Implementace v `sklearn` se chová vlastně ke všem příznakům jako ke spojitým.
- Dovede si tak celkem dobře poradit s ordinálními příznaky.
- Naopak k nominálním se chová poněkud nešťastně: V případě rovnosti informačního zisku si `sklearn.tree.DecisionTreeClassifier` vybírá náhodně. Algoritmus se tak může chovat nedeterministicky, přestože k tomu není žádný viditelný důvod.

Rozhodovací stromy pro regresi

Pokud bychom chtěli stejně jako s diskrétní pracovat se **spojitou vysvětlovanou proměnnou**, musíme vyřešit dva problémy:

1. Jak bude naučený strom určovat *predikovanou* hodnotu vysvětlované proměnné?
   - V případě diskrétní proměnné se rozhodovalo hlasováním v rámci listu: Skončila-li cesta stromem pro datový bod v listu, ve kterém z trénovacích dat skončilo nejvíce bodů s hodnotou $Y = x$, rozhodnutí daného listu bylo $Y = x$.
   - V případě spojité proměnné Y (např. věk, teplota, cena, doba trvání, ...) se může snadno stát, že každý z trénovacích bodů má jinou hodnotu.

Rozhodovací stromy pro regresi

Pokud bychom chtěli stejně jako s diskrétní pracovat se **spojitou vysvětlovanou proměnnou**, musíme vyřešit dva problémy:

- Jak bude naučený strom určovat *predikovanou* hodnotu vysvětlované proměnné?
  - V případě diskrétní proměnné se rozhodovalo hlasováním v rámci listu: Skončila-li cesta stromem pro datový bod v listu, ve kterém z trénovacích dat skončilo nejvíce bodů s hodnotou $Y = x$, rozhodnutí daného listu bylo $Y = x$.
  - V případě spojité proměnné Y (např. věk, teplota, cena, doba trvání, ...) se může snadno stát, že každý z trénovacích bodů má jinou hodnotu.

- Jaké kritérium použijeme v hladovém algoritmu pro výběr nejlepšího příznaku k dělení?
  - V případě diskrétní proměnné jsme používali *entropii* resp. *gini index*.
  - Ty jsou ale v případě spojité proměnné nepoužitelné (rozmyslete si, jak by to vypadalo, zejm. jaké hodnoty by měla čísla p_i).
  - Pro spojitou vysvětlovanou proměnnou budeme muset najít jiné kritérium.

Jak strom určuje své rozhodnutí

- Předpokládejme, že už máme naučený strom a chceme najít predikovanou hodnotu Y pro nějaký datový bod s příznaky $X_0, X_1, \ldots, X_{p-1}$.
- Pomocí rozhodovacích pravidel v daném stromu a hodnot příznaků $X_0, X_1, \ldots, X_{p-1}$ se dostaneme do listu, kde při učení „skončilo“ šest datových bodů z trénovacího množiny dat.
- Předpokládejme, že vysvětlovaná proměnná Y pro těchto šest bodů měla hodnoty

$$\{10, 15, 20, 25, 30, 35\}.$$

- Jaké má být tedy rozhodnutí stromu pro body, které skončí v tomto listu?

Jak strom určuje své rozhodnutí

- Předpokládejme, že už máme naučený strom a chceme najít predikovanou hodnotu Y pro nějaký datový bod s příznaky $X_0, X_1, \ldots, X_{p-1}$.
- Pomocí rozhodovacích pravidel v daném stromu a hodnot příznaků $X_0, X_1, \ldots, X_{p-1}$ se dostaneme do listu, kde při učení „skončilo“ šest datových bodů z trénovacího množiny dat.
- Předpokládejme, že vysvětlovaná proměnná Y pro těchto šest bodů měla hodnoty

$$\{10, 15, 20, 25, 30, 35\}.$$

- Jaké má být tedy rozhodnutí stromu pro body, které skončí v tomto listu?
- Obvykle se bere **průměr hodnot z listu**, v našem případě tedy 22.5.

Co použít místo entropie?

- V případě klasifikace se stromy rozhodovaly pomocí „hlasování“ v rámci jednotlivých listů. Minimalizace entropie měla zaručit, aby toto hlasování mělo co nejjednoznačnějšího vítěze.
- V případě regrese se strom rozhoduje tak, že v rámci listu spočítá průměr.
- Naší snahou by tedy mělo být, aby **hodnoty vysvětlované v rámci listu byly co nejblíže střední hodnoty**. Jak toto měřit?

[1] Podobně jako entropie, je i rozptyl a MSE definován pro náhodnou veličinu. My ale máme jen data, takže nepočítáme rozptyl ani MSE, ale pouze jejich odhad.

Co použít místo entropie?

- V případě klasifikace se stromy rozhodovaly pomocí „hlasování“ v rámci jednotlivých listů. Minimalizace entropie měla zaručit, aby toto hlasování mělo co nejjednoznačnějšího vítěze.
- V případě regrese se strom rozhoduje tak, že v rámci listu spočítá průměr.
- Naší snahou by tedy mělo být, aby **hodnoty vysvětlované v rámci listu byly co nejblíže střední hodnoty**. Jak toto měřit?
- Známou mírou „odchylky od střední hodnoty“ je (výběrový) **MSE = mean squared error**, což je skoro stejná veličina jako (výběrový) **rozptyl**[1] (angl. **sample variance**).

[1] Podobně jako entropie, je i rozptyl a MSE definován pro náhodnou veličinu. My ale máme jen data, takže nepočítáme rozptyl ani MSE, ale pouze jejich odhad.

Co použít místo entropie? MSE!

- Představme si, že máme data s hodnotami vysvětlované proměnné

$$\{Y_0 = 10, Y_1 = 15, Y_2 = 20, Y_3 = 25, Y_4 = 30, Y_5 = 35\}.$$

- Průměr (tj. odhad střední hodnoty) této množiny je $\overline{Y} = 22.5$.

Co použít místo entropie? MSE!

- Představme si, že máme data s hodnotami vysvětlované proměnné

$$\{Y_0 = 10, Y_1 = 15, Y_2 = 20, Y_3 = 25, Y_4 = 30, Y_5 = 35\}.$$

- Průměr (tj. odhad střední hodnoty) této množiny je $\overline{Y} = 22.5$.
- Odhad MSE je číslo $MSE(\mathbf{Y}) = (Y_0, Y_1, \ldots, Y_{N-1})$, což není nic jiného, než aritmetický průměr čtverců vzdáleností od průměru:

$$MSE(\mathbf{Y}) = \frac{1}{N} \sum_{j=0}^{N-1} (Y_i - \overline{Y})^2$$

pro naše data tedy

$$MSE((10, 15, 20, 25, 30, 35)) = \frac{1}{6}((10 - 22.5)^2 + (15 - 22.5)^2 +$$
$$+(20 - 22.5)^2 + (25 - 22.5)^2 + (30 - 22.5)^2 + (35 - 22.5)^2) = 72.9.$$

Algoritmus CART

- Hladovému algoritmu, ve kterém se vrcholy rozhodovacího stromu volí tak, aby se minimalizovalo MSE, se říká **CART = Classification and Regression Trees**.

- Funguje stejně jako algoritmus ID3 představený dříve: kvalitu rozdělení množiny $\mathcal{D}$ na podmnožiny $\mathcal{D}_L$ a $\mathcal{D}_R$ ale namísto entropie

$$H(\mathcal{D}) - t_L H(\mathcal{D}_L) - t_R H(\mathcal{D}_R)$$

používá MSE

$$\mathrm{MSE}(\mathcal{D}) - t_L \mathrm{MSE}(\mathcal{D}_L) - t_R \mathrm{MSE}(\mathcal{D}_R),$$

kde $t_L = \frac{\#\mathcal{D}_L}{\#\mathcal{D}}$ a $t_R = \frac{\#\mathcal{D}_R}{\#\mathcal{D}}$ a $\mathrm{MSE}(\mathcal{D})$ je MSE spočítané pro hodnoty vysvětlované proměnné Y pro všechny body z $\mathcal{D}$.

Poznámky

- Místo MSE lze používat také MAE = Mean Absolute Error:

$$\mathrm{MAE}(\mathbf{Y}) = \frac{1}{N} \sum_{j=0}^{N-1} |Y_i - \overline{Y}|,$$

  který místo čtverce vzdálenosti používá absolutní hodnotu.

Poznámky

- Místo MSE lze používat také MAE = Mean Absolute Error:

$$\mathrm{MAE}(\mathbf{Y}) = \frac{1}{N} \sum_{j=0}^{N-1} |Y_i - \overline{Y}|,$$

  který místo čtverce vzdálenosti používá absolutní hodnotu.

- Rozhodovací stromy mají mnoho výhod:
  - Nenáročnost na přípravu dat: poradí si s kategoriálními i spojitými příznaky, s chybějícími hodnotami, atp.
  - Jsou jednoduché a srozumitelné a učení je relativně rychlé.
  - Jsou dobře interpretovatelné a jejich rozhodnutí lze snadno rozklíčovat.

Poznámky

- Místo MSE lze používat také MAE = Mean Absolute Error:

$$\mathrm{MAE}(\mathbf{Y}) = \frac{1}{N} \sum_{j=0}^{N-1} |Y_i - \overline{Y}|,$$

  který místo čtverce vzdálenosti používá absolutní hodnotu.

- Rozhodovací stromy mají mnoho výhod:
  - Nenáročnost na přípravu dat: poradí si s kategoriálními i spojitými příznaky, s chybějícími hodnotami, atp.
  - Jsou jednoduché a srozumitelné a učení je relativně rychlé.
  - Jsou dobře interpretovatelné a jejich rozhodnutí lze snadno rozklíčovat.

- Ale mají samozřejmě i nevýhody:
  - Jsou *nerobustní*: i drobná změna v trénovacích datech může znamenat zásadní změnu struktury výsledného stromu.
  - Většina implementací podporuje pouze binární stromy.
  - Najít optimální strom je NP-úplný problém.
  - Je snadné rozhodovací stromy přeučit.