

Kapitola 2

Rozhodovací stromy

V této kapitole se seznámíte s principy rozhodovacích stromů a jejich implementací pomocí knihovny `scikit-learn`. Naučíte se, jak vytvářet, trénovat a upravovat chování rozhodovacího stromu prostřednictvím dostupných parametrů. Důraz bude kladen nejen na praktické využití, ale i na teoretické pochopení rozhodovacího mechanismu modelu.

Cíle kapitoly (po přečtení této kapitoly budete schopni)

- C1.1. Vysvětlit princip fungování rozhodovacích stromů a jejich základní terminologii (kořen, uzel, větev, list).
- C1.2. Popsat, jak probíhá tvorba rozhodovacího stromu na základě trénovacích dat.
- C1.3. Vysvětlit význam metriky *information gain* a rozdíl mezi entropií a Giniho indexem.
- C1.4. Implementovat rozhodovací strom v knihovně `scikit-learn`.
- C1.5. Použít metody vizualizace a interpretace natrénovaného stromu.
- C1.6. Vysvětlit význam vybraných hyperparametrů.
- C1.7. Ladit parametry pomocí metod `GridSearchCV` a `RandomizedSearchCV`.
- C1.8. Uvést výhody a nevýhody rozhodovacích stromů a jejich rozšíření (Random Forest, Gradient Boosted Trees)

Klíčové myšlenky (po přečtení této kapitoly byste si měli pamatovat)

- KM1.1. Rozhodovací strom je hierarchická struktura otázek a odpovědí vedoucí k výsledku.
- KM1.2. Kvalita dělení se měří pomocí entropie nebo Giniho indexu.
- KM1.3. Hyperparametry zásadně ovlivňují složitost a schopnost modelu generalizovat.
- KM1.4. Parametr `max_depth` omezuje hloubku stromu a brání přeučení.
- KM1.5. Parametr `min_samples_leaf` a `min_samples_split` určují, kdy se ještě provádí další dělení uzlů.
- KM1.6. Ladění hyperparametrů probíhá kombinací parametrů, ne izolovaně.
- KM1.7. Rozhodovací stromy se dají velmi dobře číst a vysvětlovat, ale pokud se použijí samostatně, mají často nestálé výsledky – mohou se hodně měnit podle konkrétních trénovacích dat. Proto se v praxi často spojuje více stromů dohromady do tzv. souborových (ansámblových) metod, které dávají stabilnější a spolehlivější predikce.

Nové příkazy (po přečtení kapitoly se seznámím s těmito novými příkazy)

```
DecisionTreeClassifier, DecisionTreeRegressor, fit, predict, plot_tree,
export_graphviz, classification_report, accuracy_score, GridSearchCV,
RandomizedSearchCV, train_test_split
```

2.1. Rozhodovací stromy

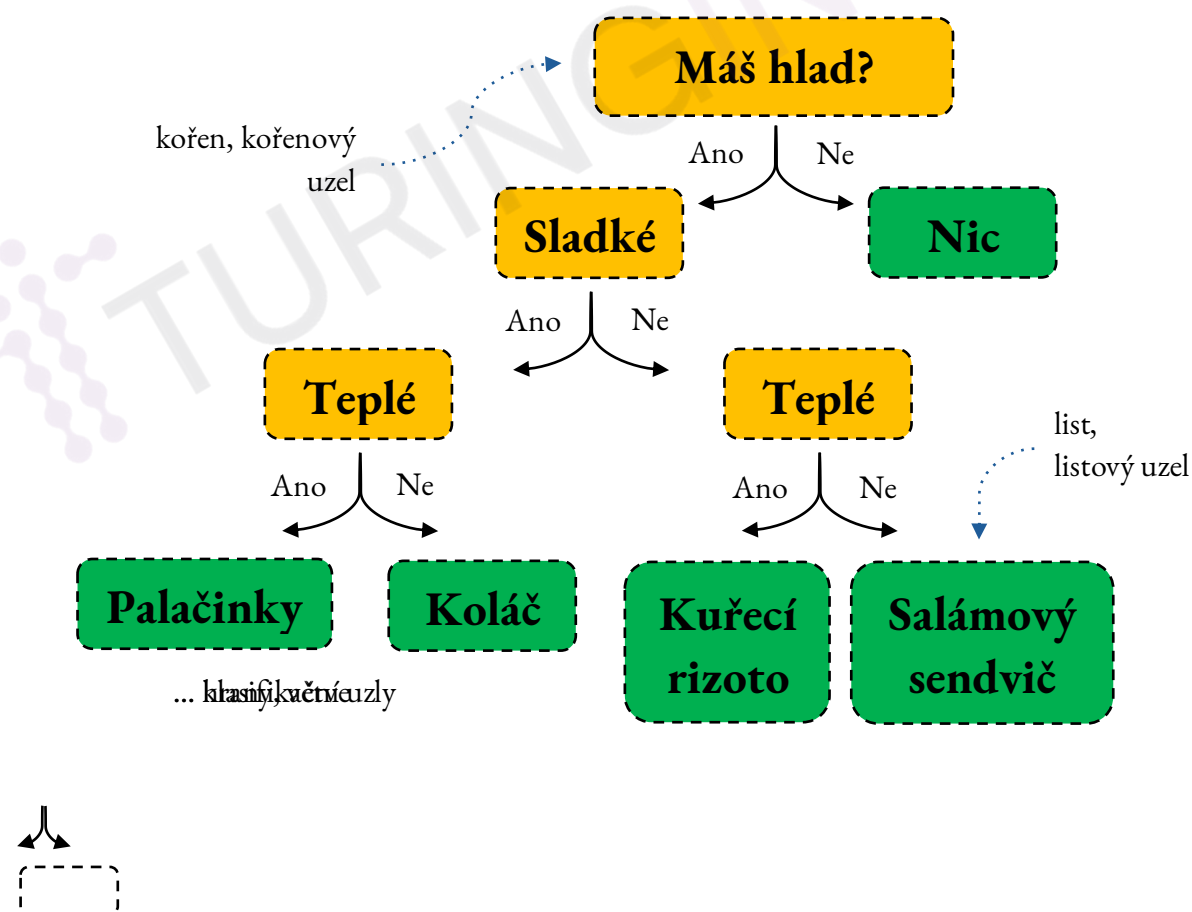
Rozhodovací strom je model používaný ve strojovém učení, který se rozhoduje na základě postupného kladení jednoduchých otázek o datech. Představit si ho můžete jako schéma, které začíná jedním hlavním rozhodnutím a větví se podle odpovědí na jednotlivé otázky, až dojde ke konečnému výsledku.

Každý **uzel** stromu obsahuje podmínku na nějaký vstupní atribut dat (například „věk“, „výška“ apod.), **větve** odpovídají různým odpovědím (například „menší než 30“, „větší než 180 cm“ apod.) a **listy** obsahují výsledek celkového rozhodnutí. Strom se při trénování vytváří tak, aby co nejlépe oddělil jednotlivé **třídy** (například „mladý/starý“, „malý/vysoký“ apod.) podle dostupných dat.

Ačkoli samotné stromy nemusí vždy poskytovat nejlepší přesnost, dají se snadno rozšířit do výkonnějších metod, kterými mohou být například „náhodný les“ (anglicky „random forest“) nebo „gradientní rozhodovací strom“ (anglicky „gradient boosted decision tree“), které různými způsoby kombinují více stromů dohromady, a tím zvyšují kvalitu predikce.

Praktický příklad pro pochopení

Každý to občas zná – máme hlad, ale nevíme, co přesně si k jídlu dát. Sladké/slané, teplé/studené, máme čas si něco uvařit, nebo chceme něco rychlého – toto jsou všechno vlastnosti, které budou rozhodovat o našem výběru jídla. V tomto příkladu si ukážeme, jak rozhodovací strom pomůže s výběrem jídla na základě několika jednoduchých otázek.



Když vytváříme rozhodovací strom, nejprve mu dáme **trénovací data** – tedy situace, ve kterých jsou známé vstupní veličiny a zároveň taktéž výstupní rozhodnutí na nich závislá.

Tabulka 1: Vstupní hodnoty (atributy) a jejich kombinace s jejich odpovídajícími výslednými rozhodnutími.

Hlad	Sladké	Teplé	Výsledek rozhodnutí
1	0	1	Kuřecí rizoto
1	0	0	Salámový sendvič
1	1	1	Palačinky
1	1	0	Koláč
0	–	–	Nic

Algoritmus pro tvorbu rozhodovacího stromu se snaží najít, který z dostupných vstupních údajů (atributů, viz Tabulka 1, „hlad“, „sladké“ a „teplé“) nejvíce pomáhá rozdělit záznamy do správných kategorií podle konečného výsledku – jinými slovy algoritmus zjišťuje, který vstupní údaj má na výsledné rozhodnutí (výsledek rozhodnutí, viz Tabulka 1, „kuřecí rizoto“, „salámový sendvič“, „palačinky“, „koláč“ a „nic“) největší vliv. K tomu algoritmus obvykle používá metriku zvanou informační zisk („information gain“), která je většinou založena na jednom ze dvou ukazatelů – entropii („entropy“) nebo Giniho indexu („gini index“). Tvorba rozhodovacího stromu lze obecně popsat následovně:

1) výběr nejlepšího atributu pro první rozhodnutí:

Na začátku algoritmus projde všechny vstupní atributy a u každého spočítá, jak dobře lze podle něj rozdělit (činnost nazývaná jako „split“ či „rozdělení uzlu“) všechny případy do co nejjednodušších tříd dle výsledku rozhodnutí. Poté algoritmus vybere ten atribut, který má největší vliv na správné rozdělení (klasifikaci, zjištění výsledku rozhodnutí) – tedy ten, který přináší nejvyšší informační zisk. V našem příkladu by to byl pravděpodobně atribut „hlad“, protože pokud platí „hlad = 0“, všechny případy vedou k výsledku rozhodnutí „nic“, takže už není potřeba žádné další rozhodování. Tento atribut je tedy schopen v prvním kroku poskytnout jasnou a jednoznačnou informaci vedoucí k určení výsledku rozhodnutí.

2) rozdělení stromu na větve:

Pro odpověď „hlad = 1“ je však nutné rozhodovat dále podle dalších atributů. Algoritmus vybere další nejvhodnější atribut (např. „sladké“) a pokračuje v dělení, dokud:

- nevzniknou tzv. **listy** se zcela jednoznačným výsledkem rozhodnutí, nebo
- už nejsou k dispozici žádné další atributy.

3) výsledná struktura stromu a použití stromu:

Jakmile je rozhodovací strom vytvořen, může být použit na nové situace, kde jsou známy hodnoty vstupních atributů, ale není známý výsledek rozhodnutí. Strom se „prochází“ od jeho kořene (první uzel) po větvích dle jednotlivých odpovědí na vybrané otázky až do listů, které obsahují jednoznačný výsledek rozhodnutí.

V tomto příkladě se však pracovalo s binárními, a hlavně diskretními vstupními veličinami, což jsou atributy, které mohou nabývat pouze dvou diskretních hodnot – například hlad = ano/ne. To

znamená, že všechny možné kombinace vstupních hodnot tvoří pevně daný a omezený počet možných výsledků rozhodnutí, v našem případě pět. Pokud trénovací data pokrývají všechny tyto kombinace, rozhodovací strom se vlastně pouze naučí přiřadit konkrétní výsledek ke každé kombinaci bez potřeby nějaké hlubší generalizace nebo učení se novým situacím. V praxi to znamená, že model neumí „vymyslet“ odpověď na situace, které v tréninku neviděl, protože žádné jiné kombinace neexistují. Kdyby došlo tedy k použití tohoto modelu například na našeho kamaráda, vypadalo by to následovně:

1. nejdříve by došlo k dotazům na jednotlivé otázky, které jsou obsaženy v našem rozhodovacím stromu,
2. odpovědi našeho kamaráda by byly zapsány a
3. nakonec by došlo k projití vytvořeného rozhodovacího stromu, který by na základě odpovědí přiřadil rozhodnutí, které by představovalo výběr jídla.

Nová data obsahující odpovědi od našeho kamaráda by tedy mohla vypadat následovně:

Hlad	Sladké	Teplé
1	0	0

Tato kombinace odpovědí by tedy při aplikaci našeho rozhodovacího stromového modelu vedla k odpovědi:

Salámový sendvič

Pro možnost skutečné generalizace a schopnost rozhodovací strom využívat i pro zcela nové, dosud neviděné vstupy, je vhodné použít vlastnosti, které mohou nabývat více než dvou hodnot – rozdělení tedy není binární a diskrétní, ale představuje spíše spojitý interval. Místo pouhého „ano/ne“ mohou data obsahovat například číselnou škálu míry hladu pohybující se v intervalu $\langle 0, 10 \rangle$ aj. V takovém případě už existuje mnohem více kombinací a model má možnost nalézt vhodné rozhodnutí i pro kombinace, které v tréninku neviděl přímo, protože dokáže vyhodnotit podobnost nebo trend v datech. Trénovací data by tedy mohla vypadat následovně:

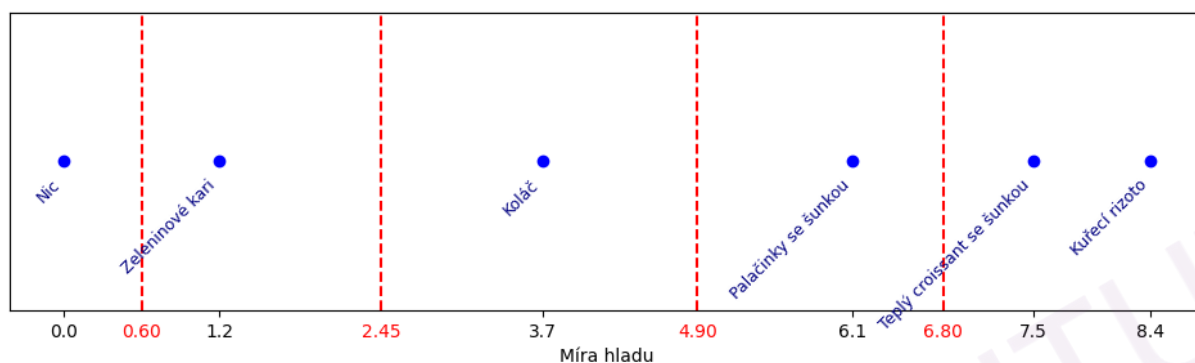
Tabulka 2: „Spojité“ vstupní hodnoty (atributy) a jejich kombinace s jejich odpovídajícími výslednými rozhodnutími.

Míra hladu	Sladké	Teplé	Výsledek rozhodnutí
1.2	0	1	Zeleninové kari
8.4	0	1	Kuřecí rizoto
3.7	1	0	Koláč
6.1	1	1	Palačinky se šunkou
0.0	0	0	Nic
7.5	1	1	Teplý croissant se šunkou

Model se na základě trénovacích dat naučí optimální prahové hodnoty pro vlastnost „míra hladu“, které efektivně rozdělují prostor vstupních hodnot na intervaly s odlišnými predikcemi. Díky tomu není nutné, aby nové vstupní hodnoty přesně odpovídaly hodnotám v trénovací sadě – model dokáže generalizovat a rozhodovat i pro hodnoty míry hladu ležící mezi trénovacími body na základě

vyhodnocení intervalu, do kterého spadají, a následném přiřazení jejich odpovídajícího výstupu. Rozdělení prostoru vstupních hodnot na základě vlastnosti míry hladu si můžeme zjednodušene (jelikož výsledek rozhodnutí je závislý ještě na dalších veličinách) představit následovně:

Při použití sofistikovanějšího modelu by nová data obsahující nové odpovědi mohla vypadat následovně:



Míra hladu	Sladké	Teplé
3.2	0	0

Tato kombinace odpovědí by tedy při aplikaci našeho nového rozhodovacího stromového modelu vedla k odpovědi:

Koláč

Všimněte si, že i navzdory tomu, že rys „sladké“ nabýval hodnoty 0, výsledkem byl z důvodu rozdělení prostoru vstupních rysů (viz Obrázek 1) stále „koláč“.

2.2. Využití knihovny scikit-learn

Knihovna **scikit-learn** je určená pro praktické použití metod strojového učení v jazyce Python. Umožňuje jednoduchým způsobem vytvářet, trénovat a vyhodnocovat modely bez nutnosti implementace jednotlivých algoritmů od základu. Pro práci s rozhodovacími stromy poskytuje připravené třídy jako *DecisionTreeClassifier* pro klasifikaci a *DecisionTreeRegressor* pro regresi, které pokrývají celý proces od učení modelu po predikci výsledků.

Kromě samotného natrénování modelu umožňuje knihovna také snadno zobrazit rozhodovací strom pomocí funkce `plot_tree`. Dále nabízí nástroje pro nastavování parametrů modelu, hodnocení jeho přesnosti a propojení s dalšími kroky, jako je příprava a úprava vstupních dat.

Seznámení se s API knihovny scikit-learn

```
#####
#                                     0. Import knihoven
#####
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

```

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import load_breast_cancer

#####
# 1. DATA - vstupy a cílové třídy #####
dataset = load_breast_cancer()
X = dataset.data
y = dataset.target

#####
# 2. Rozdělení na trénink/test #####
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3,
random_state=1)
print(X_train.shape)
print(X_test.shape)

#####
# 3. Model - rozhodovací strom #####
clf = DecisionTreeClassifier(
    criterion="entropy",
    max_depth=4,
    min_samples_leaf=1,
    random_state=42,
    max_features=None,
    min_samples_split=2,
    max_leaf_nodes=None,
    splitter="best"
)

clf.fit(X_train, y_train)

#####
# 4. Vyhodnocení #####
y_pred = clf.predict(X_test)
print("Přesnost:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

#####
# 5. Vizualizace stromu #####
plt.figure(figsize=(14, 8))
plot_tree(
    clf,
    feature_names = dataset.feature_names,
    class_names=[str(c) for c in clf.classes_],
    filled=True,
    rounded=True
)
plt.title("Rozhodovací strom")
plt.show()

```

```
import graphviz
from sklearn.tree import export_text, export_graphviz

exportovany_strom = export_graphviz(
    clf,
    feature_names = dataset.feature_names,
    class_names = dataset.target_names,
    filled=True,
    rounded=True,
    leaves_parallel=True
)

vizualni_graf = graphviz.Source(exportovany_strom)
vizualni_graf.render("stromecek")
```

0. Import knihoven:

Importuje potřebné metody zvolených knihoven – v tomto případě **rozhodovací strom** *DecisionTreeClassifier*, dále metody *plot_tree* pro vizualizaci natrénovaného rozhodovacího stromu, *train_test_split*, která umožňuje rozdělení původního datasetu na trénovací a testovací sadu, a *accuracy_score* sloužící k výpočtu přesnosti konečné klasifikace a *classification_report* poskytující podrobný přehled o výkonu modelu.

1. Vytvoření původního datasetu, který slouží k trénování základního modelu a k testování již natrénovaného modelu.
2. Rozdělení původního datasetu na trénovací a testovací sadu dat.
3. Definice modelu (rozhodovacího stromu), nastavení jeho základních parametrů:
 - a. *criterion="entropy"* ... definice kritéria, dle kterého dochází k výpočtu tzv. „information gain“ hodnoty, která při trénování rozhoduje o tvorbě jednotlivých hranic rozdělujících prostor vstupních hodnot a umožňujících pozdější klasifikaci, dále může nabývat hodnoty „*gini*“ a „*log_loss*“;
 - b. *max_depth* ... definice hodnoty maximální hloubky rozhodovacího stromu, která určuje, kolik úrovní větvení rozhodovací strom může mít, čím větší hodnota, tím složitější a rozvětvenější strom, menší hodnota naopak omezuje tzv. „přeučení“ neboli „overfitting“;
 - c. *min_samples_leaf*... definice hodnoty minimálního počtu vzorků v každém listovém uzlu rozhodovacího stromu, vyšší hodnota tohoto parametru zabraňuje tvorbě příliš malých a specifických větví tím, že stanovuje minimální počet vzorků, které musí být v každém listu stromu, a zabraňuje tím tak tzv. přeučení neboli „overfitting“;
 - d. *random_state* ... definice hodnoty, která nastavuje pevný tzv. „seed“ pro generátor náhodných čísel, aby byla zaručena opakovatelnost výsledků, díky čemuž je možné při opakovaném spuštění algoritmu získat stejné výsledky – lze použít libovolné číslo;
 - e. *max_features* ... definice počtu vstupních vlastností (*features*), které se při hledání nejlepšího rozdělení v každém uzlu zvažují; může být *int* (představuje konkrétní počet vlastností), *float* (představuje zlomek z celkového počtu vlastností – např. 0.5

představuje 50 % všech vlastností), *"auto"* a *"sqrt"* (obě představují totéž, a tedy že se počet vlastností rovná odmocnině z celkového počtu vlastností), *"log2"* (počet vlastností se rovná logaritmu o základu 2 z celkového počtu vlastností) nebo *None* (použijí se všechny vlastnosti), pomáhá regulovat náhodnost modelu a zabraňuje přeučení;

- f. *min_samples_split* ... definice minimálního počtu vzorků, které musí být přítomny v uzlu, aby mohl být dále rozdělen, vyšší hodnota omezuje tvorbu příliš složitých a detailních rozhodovacích stromů, a snižuje tak riziko přeučení;
- g. *splitter* ... představuje strategii výběru dělicího bodu (dělicí roviny, nadroviny apod.) v rozhodovacím uzlu – může nabývat hodnot *"best"* (vybere nejlepší dělení podle zvoleného kritéria) nebo *"random"* (vybere náhodné rozdělení v rozhodovacím uzlu z omezené množiny dostatečně kvalitních kandidátů, tzv. náhodný „split“ (náhodné „rozdělení uzlu“) – ignoruje tedy nejlepší hodnotu tzv. „information gain“, vybere několik možných rozdělení, které disponují vysokými hodnotami „information gain“ a z nich vybere již náhodně), použití hodnoty *"random"* může zlepšit generalizaci modelu

a jeho natrénování pomocí metody *fit(X_train, y_train)*.

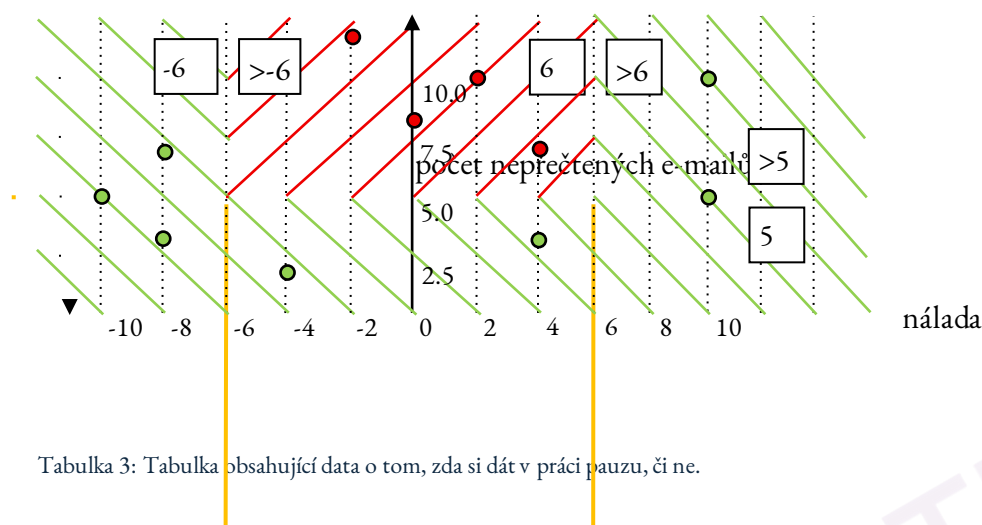
- 4. Vyhodnocení modelu na testovacích datech pomocí metody *predict(X_test)* a vypsání základních metrik pro vyhodnocení jeho přesnosti pomocí metod *accuracy_score(y_test, y_pred)* a *classification_report(y_test, y_pred)*.
- 5. Vizualizace rozhodovacího stromu pomocí metody *plot_tree* a nastavení jejích základních parametrů:
 - a. *clf* ... vložení objektu reprezentujícího dříve vytvořený rozhodovací strom;
 - b. *feature_names* ... definice seznamu názvů vstupních proměnných (vlastností) z původního datasetu, zobrazených v uzlech rozhodovacího stromu;
 - c. *class_names* ... definice seznamu názvu tříd, které rozhodovací strom klasifikuje, zobrazených v „listových uzlech“ stromu;
 - d. *filled* ... parametr nastavující, jestli jsou uzly barevně vyplněné podle třídy nebo hodnoty, což usnadňuje orientaci;
 - e. *rounded* ... parametr nastavující, jestli mají uzly zaoblené rohy, což zlepšuje vizuální vzhled rozhodovacího stromu.

2.3. Popis vybraných parametrů a jejich matematická implementace

Criterion

Parametr *criterion* v modelu *DecisionTreeClassifier* ve scikit-learn určuje způsob, jakým se vyhodnocuje kvalita jednotlivých dělení dat („splitů“) v uzlech rozhodovacího stromu. Jinými slovy, říká, jak bude strom vybírat nejlepší dělicí hranici (dělicí bod, dělicí rovinu, nadrovinu apod.) na základě toho, jak moc se tím daným dělením zvýší „čistota“ rozdělených skupin. Tento proces se označuje jako výpočet tzv. *information gain* (zisk informace). Princip rozdělení dat dle zvoleného

kritéria bude ukázán na následujícím příkladu týkajícím se klasifikace toho, zda si máme dát v práci pauzu, či ne.



Tabulka 3: Tabulka obsahující data o tom, zda si dát v práci pauzu, či ne.

Nálada	Počet nepřečtených e-mailů	Dát si pauzu?
-10	5	Ano
-8	3	Ano
-8	7	Ano
-4	2	Ano
-2	12	Ne
0	8	Ne
2	10	Ne
4	3	Ano
4	7	Ne
10	5	Ano
10	10	Ano

Data vyobrazená v grafu výše jsou reprezentována pomocí tabulky č. 3. Jedná se o datový soubor, který obsahuje číselné odpovědi na kvalitu nálady a počet nepřečtených e-mailů, na základě kterých se člověk může rozhodnout, jestli si dá v práci pauzu, či nedá. Rozhodovací strom si při svojí tvorbě vyčlenil hranice (viditelné v grafu oranžovou barvou), které mu pomáhají v pozdějším rozhodnutí.

Struktura tohoto stromu bude vypadat následovně:

A: entropie

$$IG = Entropy(parent) - \sum w_i \cdot Entropy(child_i),$$

kde:

$$Entropy = - \sum p_i \cdot \ln(p_i),$$

kde p_i představuje pravděpodobnost i-té třídy (skupiny).

B: Giniho index

$$IG = GINI(parent) - \sum w_i \cdot GINI(child_i),$$

kde:

$$GINI = 1 - \sum p_i^2,$$

kde p_i představuje pravděpodobnost i-té třídy (skupiny).

Pro porovnání dvou rozdělení („splitů“) X a Z bude využit ukazatel **entropie**, velikost hodnoty informačního zisku rozhodne o tom, jaké rozdělení, čili jakou hranici, si při své tvorbě rozhodovací strom vybere.

X:

Nálada ≤ -6

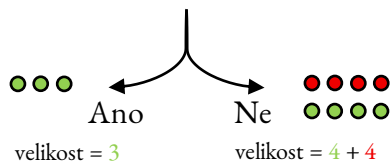
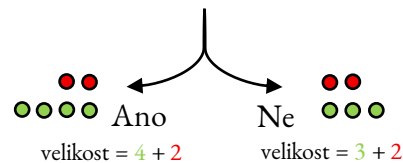


Z:

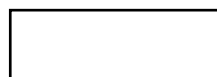
Nálada ≤ 0



velikost = 7 + 4



Obdobně pro hranici Z:



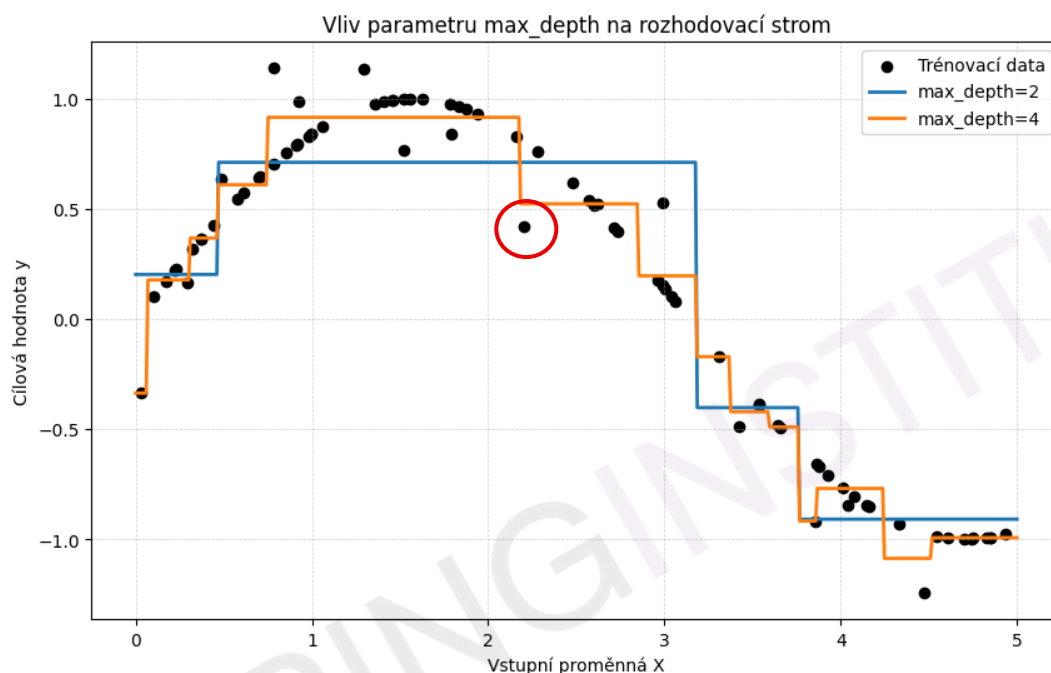
Při srovnání obou hodnot informačního zisku mezi hranicemi X a Z lze jednoznačně vidět, že hranice označená písmenem X představuje z hlediska informačního zisku značně kvalitněji zvolenou hranici, než je hranice označená písmenem Z. Při algoritmu tvorby rozhodovacího stromu dochází k výběru nejlepší (při nastavení strategie výběru hranic *splitter* = „best“) hranice (hranice „splitu“) ze všech možných hranic, které jsou vybírány ze vstupních dat (atributů). Algoritmus tak tedy při tvorbě rozhodovacího stromu postupuje od „parent“ uzlů k „child“ uzlům prostřednictvím tvorby kvalitních hranic a svůj proces ukončuje, až:

- dosáhne tzv. listů – tedy uzlů, které obsahují datové body spadající pouze do jedné (stejně) třídy, tj. 100% čistých uzlů,
- dosáhne definované maximální hloubky (v knihovně scikit-learn *max_depth* parametr),
- dosáhne uzlů obsahujících množství datových bodů, které je menší než minimální počet datových bodů v uzlu, při kterém má ještě dojít k vytvoření další hranice (hranice „splitu“), který je v knihovně scikit-learn definován parametrem *min_samples_split*.

Při využití Giniho indexu by došlo pouze k náhradě vztahu pro entropii vztahem pro Giniho index.

Max_depth

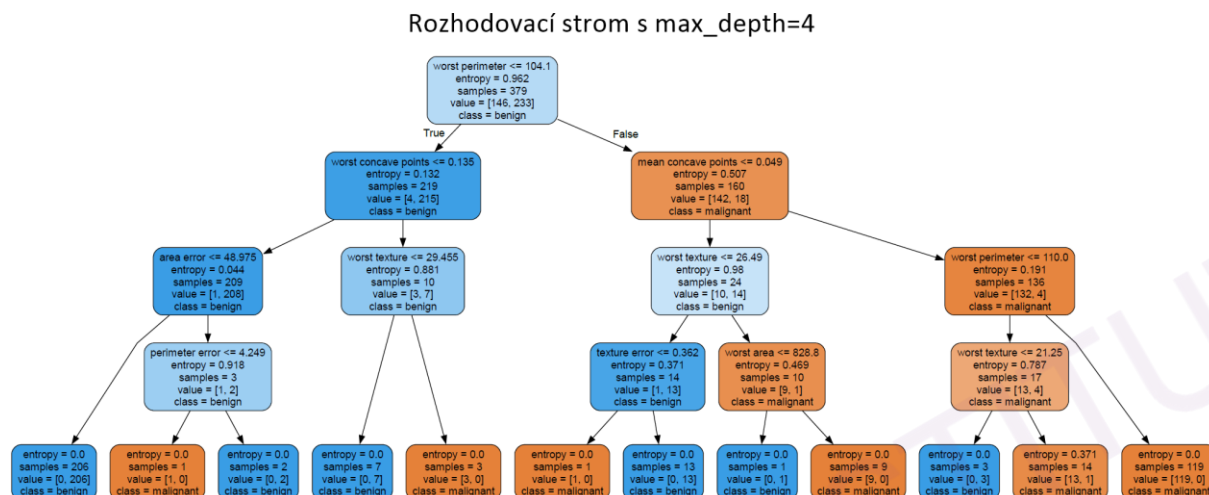
Parametr *max_depth* v modelu *DecisionTreeClassifier* ve scikit-learn určuje maximální povolenou hloubku rozhodovacího stromu, tedy nejvyšší počet úrovní od **kořene** stromu až po **listy**. Tento parametr přímo ovlivňuje složitost modelu a jeho schopnost zachytit vzory v datech. Menší hodnota *max_depth* vede k jednodušším rozhodovacím stromům, které se zaměřují pouze na nejvýraznější rozdělení dat, což může snížit riziko přeučení modelu (tzv. „overfitting“ – vysvětleno dříve), ale zároveň zvýšit riziko nedostatečného naučení modelu (tzv. „underfitting“ – vysvětleno dříve).



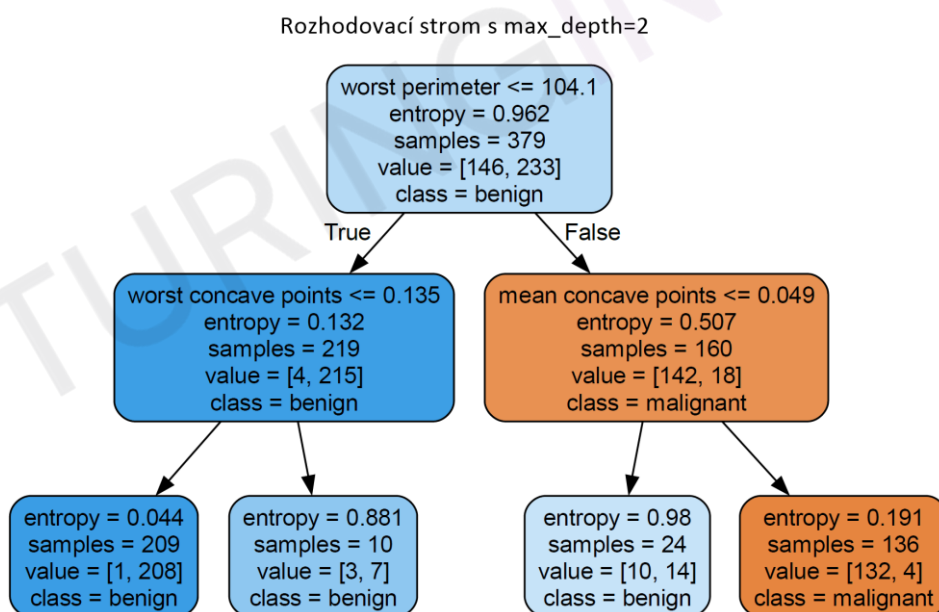
Naopak vyšší hodnota parametru *max_depth* umožní stromu provádět více rozhodovacích kroků, a tedy přesněji přizpůsobit model trénovacím datům, což však může vést k přeučení a zhoršené schopnosti generalizace modelu na nová doposud neviděná data. Nastavení parametru *max_depth* je proto důležitým nástrojem pro regulaci složitosti modelu a nalezení rovnováhy mezi přesností na trénovacích datech a výkonem na datech testovacích.

Na grafu zobrazeném na obrázku výše lze vidět přesnost modelu a její závislost na hodnotě parametru *max_depth*. Na obrázcích níže jsou přímo vyobrazeny rozhodovací stromy, u kterých lze jednoznačně vidět hodnotu tohoto parametru.

Z grafu a obrázku č. 3 lze konstatovat, že hodnota *max_depth* = 2 je příliš nízká, a dochází tak k tzv. „underfitting“, což znamená, že je vytvořený model příliš jednoduchý a nedokáže zachytit složitost testované úlohy.



Z grafu a obrázku č. 4 lze konstatovat, že hodnota *max_depth* = 4 již představuje lepší hodnotu



Obrázek SEQ Obrázek * ARABIC 3: Struktura rozhodovacího stromu, jehož velikost byla omezena parametrem *max_depth*

zvoleného parametru, avšak v určitých bodech (v červeném kroužku) lze vidět neschopnost zobecnění takto vytvořeného modelu poukazující na tzv. „overfitting“, čili přeučení modelu na trénovacích datech.

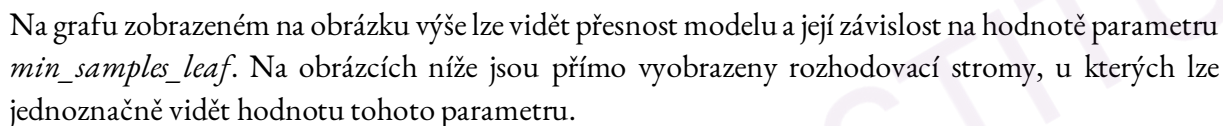
Min_samples_leaf

Parametr *min_samples_leaf* v modelu *DecisionTreeClassifier* určuje minimální počet vzorků, které musí zůstat v každém listovém uzlu poté, co strom provede rozdělení. Při tvorbě stromu tedy algoritmus nikdy neprovede takové rozdělení uzlu, pokud by v některé z nově vzniklých větví skončilo méně vzorků, než je hodnota tohoto parametru.

Tento parametr tedy nepřímo ovlivňuje hloubku a jemnost struktury stromu. Pokud je nastaven na 1 (výchozí hodnota), strom může dělit až do okamžiku, kdy vznikají listy s jediným vzorkem. Pokud je například nastaven na 10, pak strom ukončí dělení dříve, protože musí zajistit, aby v každém listu zůstalo minimálně 10 vzorků – i kdyby tyto vzorky byly „smíšené“ z různých tříd.

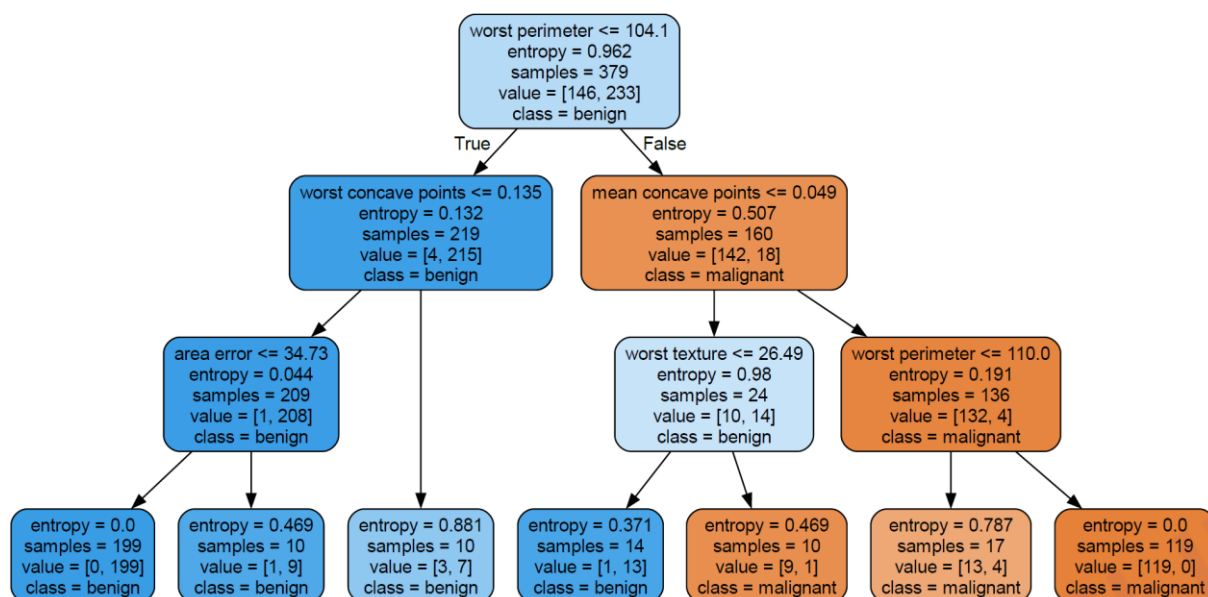
- nižší hodnota (např. 1) umožňuje velmi jemná dělení, což může vést k přeučení („overfitting“);
- vyšší hodnota (např. 10 nebo více) nutí strom dělit méně, což snižuje riziko přeučení, ale může naopak vést k nedostatečnému naučení modelu („underfitting“), protože strom nedokáže zachytit všechny detaily v datech.



[illegible]

Turing Institut · turinginstitut.cz

Rozhodovací strom s $\text{min_samples_leaf}=10$



Z grafu na obrázku č. 5 a obrázku č. 7 lze konstatovat, že hodnota $\text{min_samples_leaf} = 10$ je příliš vysoká, a dochází tak k tzv. „underfitting“, což znamená, že je vytvořený model příliš jednoduchý a nedokáže zachytit složitost testované úlohy.

Při ladění hyperparametrů se **nehledá** optimální hodnota každého parametru izolovaně, ale vždy jde o volbu celé kombinace parametrů. To znamená, že při hledání nejlepší konfigurace vstupujeme do vícerozměrného prostoru všech možných nastavení, tzv. hyperparametrického prostoru. V něm se postupně zkoumají různé varianty a vyhodnocuje se jejich vliv na výkon modelu. Tento proces je výpočetně náročnější, protože parametry mezi sebou vzájemně interagují – změna jednoho může mít odlišný efekt v závislosti na hodnotě jiného. Z toho důvodu nelze jednoduše optimalizovat každý parametr samostatně, ale je třeba uvažovat jejich společné působení.

2.4. Ladění hyperparametrů metody rozhodovacích stromů

V rámci strojového učení není často nejdůležitější pouze samotná volba algoritmu, ale také správné nastavení jeho hyperparametrů – tedy hodnot, které nelze zjistit učením z dat, ale je třeba je definovat ručně (například *max_depth*, *criterion*, *min_samples_leaf* apod.). K jejich systematickému hledání slouží dvě nejčastější metody:

1. GridSearchCV (mřížkové vyhledávání)

Metoda ladění hyperparametrů, která systematicky prochází všechny možné kombinace z předem definovaného prostoru hodnot jednotlivých hyperparametrů. Pokud například uživatel zadá tři různé hodnoty pro parametr *max_depth* a tři hodnoty pro *min_samples_leaf*, bude model natrénován a vyhodnocen pro všech devět vzniklých kombinací. Tento přístup zaručuje, že se prozkoumá celý vymezený prostor možností, a proto existuje vysoká pravděpodobnost nalezení optimální sady parametrů zvoleného modelu.

Největší výhodou GridSearch je tedy jeho úplnost – žádná kombinace nezůstane opomenuta. Na druhou stranu je však tato metoda výpočetně velmi náročná, protože počet trénovaných modelů roste s počtem testovaných hodnot exponenciálně. GridSearch je proto vhodnější pro menší množiny parametrů nebo v případech, kdy není problém využít větší výpočetní výkon.

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Dataset (X = features, y = labels)
iris = load_iris()
X, y = iris.data, iris.target

# Model
model = DecisionTreeClassifier(random_state=42)

# Prostor parametrů pro systematické testování
param_grid = {
    "max_depth": [2, 4, 6],
    "min_samples_leaf": [1, 2, 4],
    "criterion": ["gini", "entropy"]
}

# Vytvoření instance GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)

# Trénování (fit) s použitím dat X a y
grid_search.fit(X, y)

# Výpis nejlepší kombinace parametrů
print("Nejlepší parametry:", grid_search.best_params_)
```

2. RandomizedSearchCV (náhodné vyhledávání)

Metoda ladění hyperparametrů, která místo systematického procházení všech možných kombinací náhodně vybírá určité množství kombinací z předem definovaného prostoru hodnot hyperparametrů. Uživatel zde nemusí zvolit přesné hodnoty každého parametru, stačí nastavit pouze jejich konkrétní intervaly, ze kterých se hodnoty budou samy náhodně generovat (například s využitím rovnoměrného rozdělení či jiných pravděpodobnostních funkcí). Místo kompletní mřížky se tedy náhodně otestuje jen omezený počet kombinací.

Tento přístup má obrovskou výhodu v rychlosti a efektivitě – umožňuje nalézt velmi dobré, i když ne vždy globálně nejlepší řešení, a přitom výrazně snižuje výpočetní náročnost. RandomizedSearch se hodí zejména tehdy, když je prostor hyperparametrů velmi rozsáhlý nebo když je pouze omezený čas na trénování modelu.

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Dataset (X = features, y = labels)
iris = load_iris()
X, y = iris.data, iris.target

# Model
model = DecisionTreeClassifier(random_state=42)

# Náhodné rozdělení parametrů
param_dist = {
    "max_depth": randint(2, 10),           # náhodné hodnoty mezi 2 a 9
    "min_samples_leaf": randint(1, 5),     # náhodné hodnoty mezi 1 a 4
    "criterion": ["gini", "entropy"]
}

# Vytvoření instance RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_dist,
    n_iter=10,
    cv=5,
    random_state=42
)

# Trénování (fit) s použitím dat X a y
random_search.fit(X, y)

# Výpis nejlepší kombinace parametrů
print("Nejlepší parametry:", random_search.best_params_)
```

2.5. Výhody rozhodovacích stromů a knihovny scikit-learn, aplikovatelnost

Rozhodovací stromy nacházejí široké uplatnění v mnoha oblastech díky své srozumitelnosti, jednoduché interpretaci a schopnosti pracovat jak s číselnými, tak i kategoričnými daty (např. barva, typ, značka). Jejich výstupem je přehledná hierarchie rozhodnutí (tzv. stromová struktura), kterou

lze snadno vizualizovat a interpretovat i bez hlubších znalostí strojového učení. Na rozdíl od složitějších modelů (např. neuronových sítí aj.), kde je rozhodovací proces skrytý (tzv. „black-box“), rozhodovací strom poskytuje uživateli transparentní a vysvětlitelné rozhodování. Díky této vlastnosti jsou stromy využívány zejména tam, kde je potřeba nejen přesná predikce, ale i srozumitelné vysvětlení rozhodnutí – tedy v oblastech s požadavky na transparentnost a důvěryhodnost modelu.

Knihovna scikit-learn je jedna z nejpoužívanějších Python knihoven pro strojové učení. Nabízí snadno použitelnou implementaci rozhodovacích stromů prostřednictvím tříd:

- *DecisionTreeClassifier* pro klasifikaci a
- *DecisionTreeRegressor* pro regresi.

Hlavní výhody této knihovny v kontextu rozhodovacích stromů jsou:

- jednoduchá syntaxe a konzistentní rozhraní – model lze vytvořit, natrénovat a použít ve třech řádcích kódu,
- možnost vizualizace stromu,
- snadné ladění parametrů a integrace s nástroji pro hledání optimálních hodnot,
- podpora vyhodnocení výkonu modelu pomocí dostupných metrik a
- důležitost vstupních proměnných lze snadno interpretovat pomocí dostupného atributu.

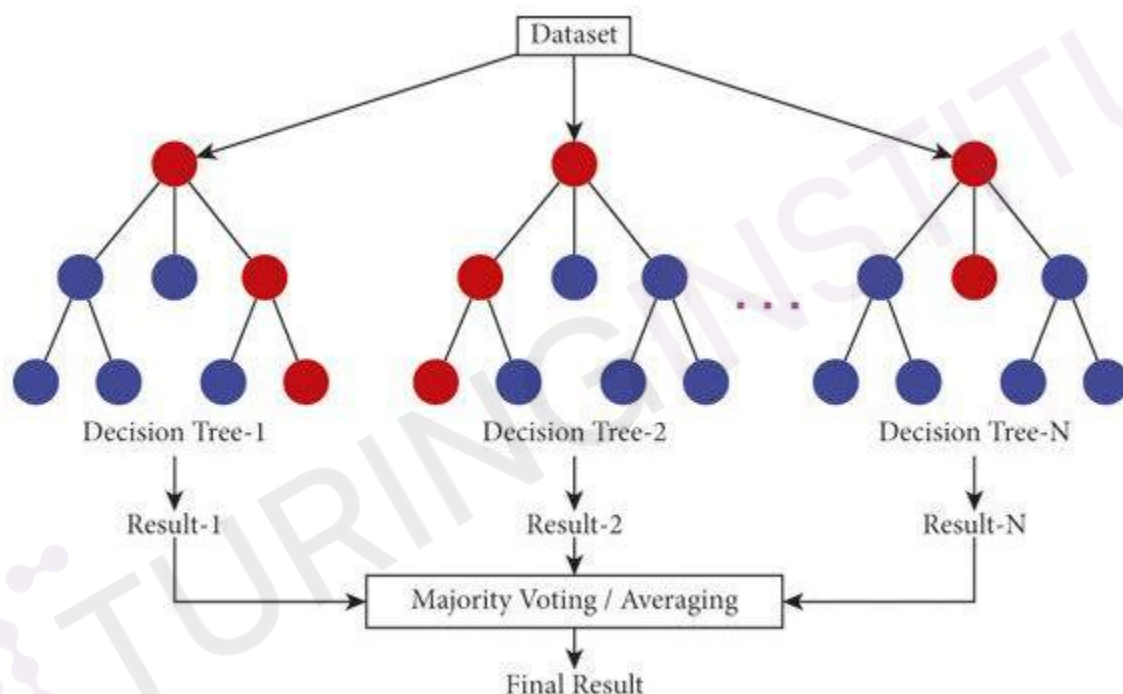
Díky těmto vlastnostem je scikit-learn velmi vhodný jak pro začátečníky, tak i pro pokročilé vývojáře a analytiku pracující s reálnými daty. Knihovna umožňuje rychle testovat modely, interpretovat výsledky a jednoduše je integrovat do širších analytických nebo softwarových systémů.



2.6. Rozšíření základní metody rozhodovacích stromů

Základní rozhodovací stromy mají omezení v podobě vysoké variance (kolísání výsledků na různých trénovacích datech) a náchylnosti k přeučení (tzv. „overfitting“ – model se příliš přizpůsobí trénovacím datům a špatně se generalizuje na nová data). Pro zlepšení přesnosti a robustnosti (odolnost vůči šumu a variabilitě vstupních dat) se proto používají metody skládající více stromů, jako jsou „Random Forest“ a „Gradient Boosted Decision Trees“.

Metoda „**Random Forest**“ funguje na principu vytváření velkého počtu nezávislých rozhodovacích stromů (tzv. „ensemble“ – soubor modelů), které jsou trénovány na náhodných podmnožinách dat a vstupních vlastností (tzv. „bagging“ neboli „bootstrap aggregating“). Výsledná predikce je pak průměrem (v případě regresních úloh) nebo většinovou hodnotou (u klasifikace) těchto jednotlivých rozhodovacích stromů. Tento přístup snižuje přeučení díky rozložení chyby mezi více modelů a zvyšuje stabilitu výsledků.



Metoda „**Gradient Boosted Decision Trees**“ vytváří stromy postupně (postupné učení), čili každý následující rozhodovací strom se zaměřuje na chyby předchozích rozhodovacích stromů tím, že minimalizuje ztrátovou funkci (funkci hodnotící chybu modelu) pomocí gradientního sestupu, což je optimalizační metoda hledající směr nejrychlejšího snižování chyby. Tímto způsobem se model učí z chyb a postupně zlepšuje přesnost. Metoda „Gradient Boosted Decision Trees“ je obvykle přesnější než metoda „Random Forest“, ale může být náchylnější k přeučení, proto vyžaduje pečlivé ladění parametrů.

Obě metody představují výkonná rozšíření klasických rozhodovacích stromů, která kombinují jednoduchost jednotlivých rozhodovacích stromů s vyšší přesností a lepší schopností generalizace na nová data.