Pokročilé metody klasifikace

Pokyny k online schůzce

1. Používejte nejlépe desktopovou aplikaci Teams.

2. Pokud nekladete dotaz nebo se neúčastníte diskuze, mějte prosím vypnutý mikrofon.

3. Pokud jen trochu můžete, mějte puštěnou kameru. Vás výraz tváře pomůže vyučujícímu :)

4. Jak můžete položit dotaz:
   a. Zapněte si mikrofon a rovnou se zeptejte.
      nebo:
   b. Napište dotaz do chatu. nebo:
   c. Použijte tlačítko zvednout ruku.
      (Po vyvolání ruku sundejte)

Agenda

- Support Vector Machines
- Rozhodovací stromy
- Ensemble Learning
- Náhodné lesy
- Gradient Boosting

Support Vector Machines

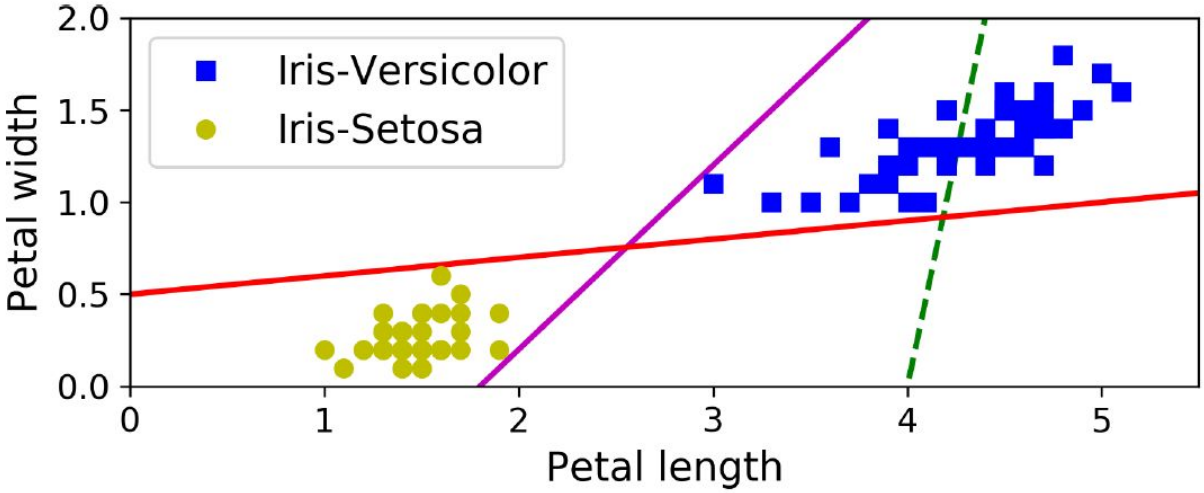
Support Vector Machines

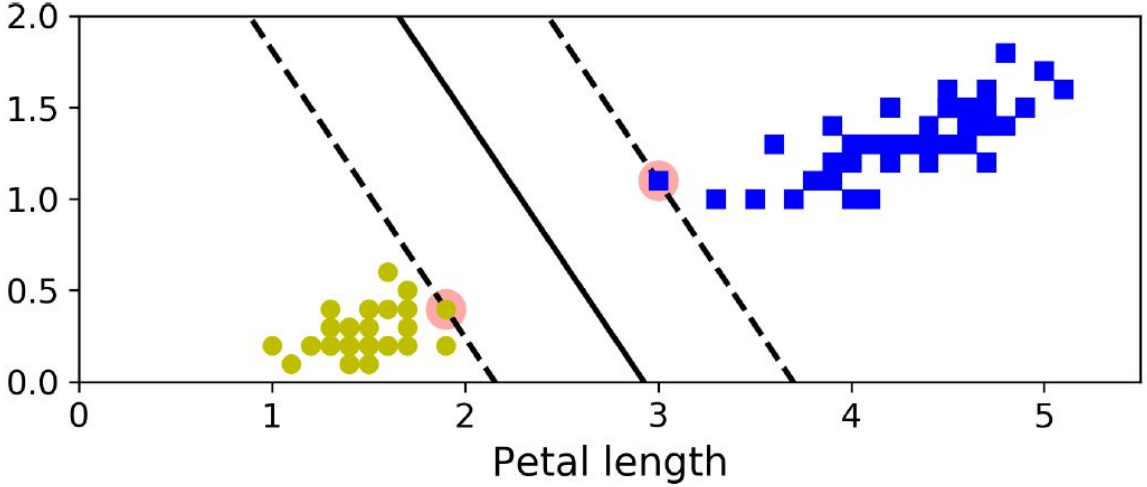
- “metoda podpůrných vektorů”
- výkonný a univerzální model, který je schopen provádět tyto úlohy:
  - lineární i nelineární klasifikace
  - regrese
  - detekce anomalit
- jeden z nejoblíbenějších modelů, zejména pro klasifikaci komplexních, ale malých (až středně velkých) datasetů
  - pro velké datasety obtížně použitelné
  - neposkytují dobré výsledky pro problémy vnímání (obraz, zvuk, ...), protože SVM je mělkou metodou vyžadující dobrou konstrukci příznaků (feature engineering), což je zejména v případě problémů vnímání obtížné a křehké a prim zde tak hrají neuronové sítě
- na Titanic datasetu měl v mém případě nejlepší výsledky

Lineární SVM

- algoritmus SVM se snaží najít ideální nadrovinu (hyperplane), která co nejlépe separuje dataset do příslušných tříd
  - nadrovina prostoru dimenze N je jeho podprostor dimenze N-1
    - ve 2D rovině je její nadrovinou jakákoliv přímka
    - ve 3D prostoru je jeho nadrovinou jakákoliv 2D rovina
  - jde tedy o to, že tato nadrovina dokáže prostor dané dimenze rozdělit na dvě části

- nejlépe viz příklad např. na datasetu Iris





Zdroj: (Géron 2019)

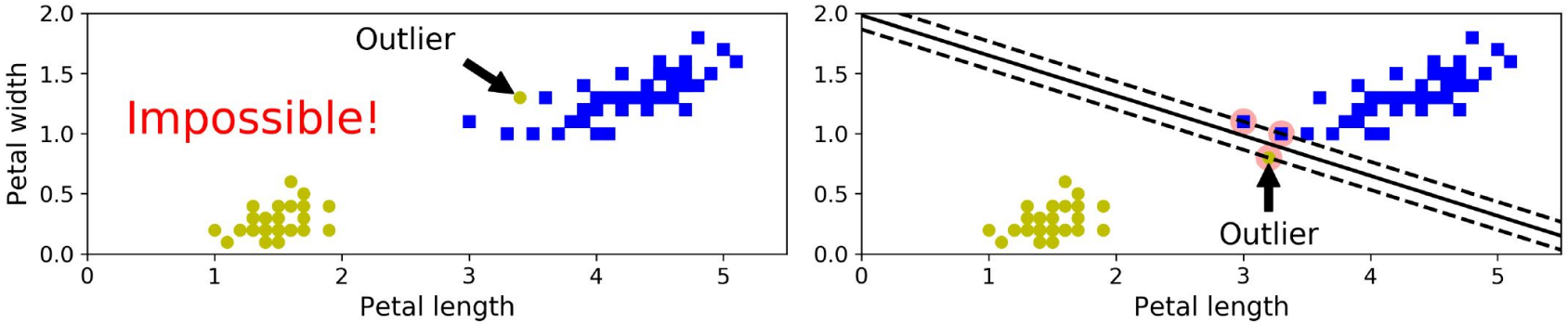
Lineární SVM

- rozhodovací hranice v případě modelu SVM se snaží být tak daleko od obou tříd, jak je to jen možné
  - “cílem je protáhnout datasetem co nejširší ulici”
  - nové instance, které nejsou na hranici “ulice”, její podobu nijak neovlivní
  - instance, které jsou na hranici “ulice” (zakroužkované), jsou nazývané podpůrné vektory (a odtud název metody)

- metoda SVM je silně závislá na škálování příznaků, musíme tedy příznaky důsledně škálovat, pokud chceme použít SVM

Hard margin vs. soft margin SVM

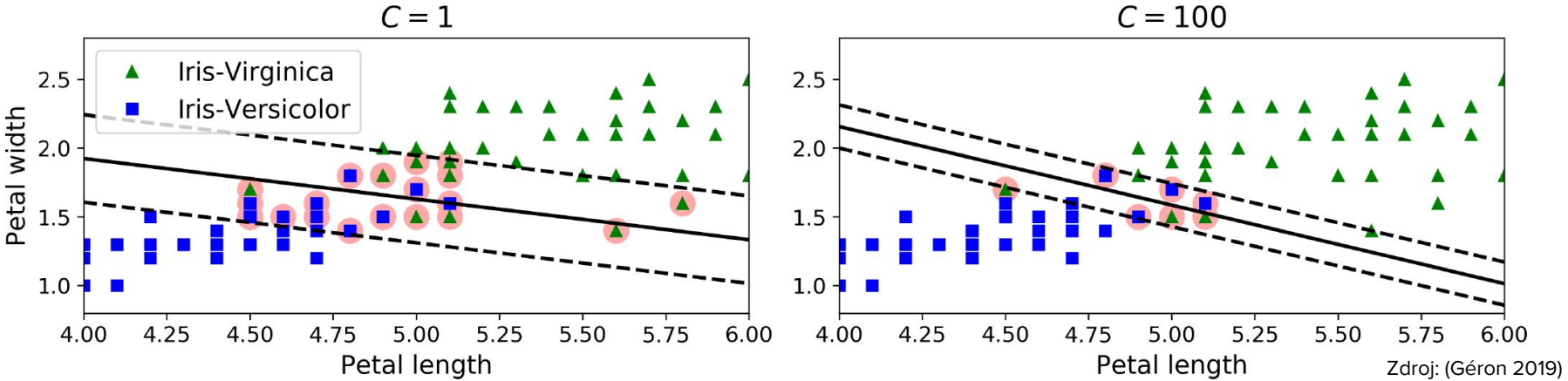
- hard margin - chceme, aby všechny instance byly mimo “ulici” (a samozřejmě na správné straně), to má však problémy:
  - dataset musí být dokonale lineárně separovatelný (což je v praxi málo kdy)
  - náchylnost na odlehlé instance



Zdroj: (Géron 2019)

Soft margin SVM

- uvedené problémy řeší tzv. soft margin - hledá se rovnováha mezi šířkou ulice a tím, jak moc jsou porušeny její hranice (instancemi, které jsou na ulici nebo dokonce na špatné straně rozhodovací hranice)
- hyperparametr C - nízké hodnoty vedou k širší ulici a více porušení hranic a naopak (nízké C lze použít jako nástroj pro snížení přeučení SVM modelu)



Zdroj: (Géron 2019)

Praktická ukázka lineárního SVM

- viz jupyter

- LinearSVC(C=1, loss=”hinge”)
  - nejrychlejší způsob, pokud si vystačíme s lineární modelem
- SVC(kernel=”linear”, C=1)
  - univerzální SVM, ale pomalejší
  - jako jediná podporuje tzv. kernel trick (viz dále)
- SGDClassifier(loss=”hinge”, alpha=1/(m*C))
  - založeno na SGD, takže vhodné pro větší datasety a online learning

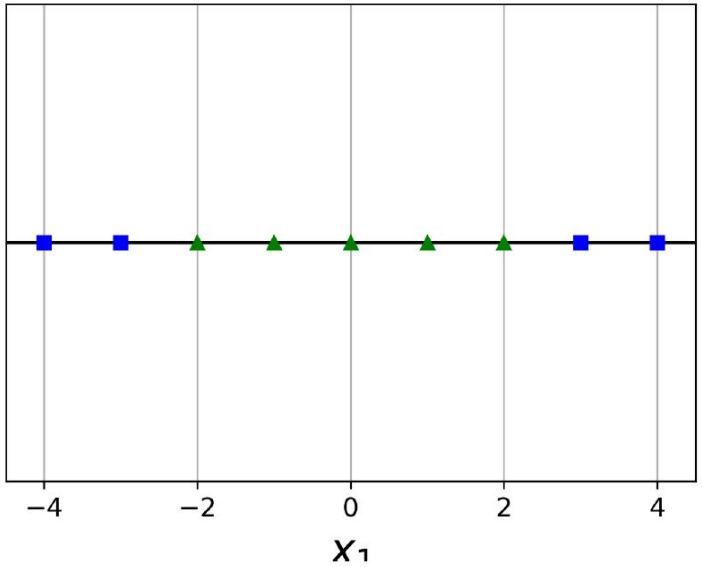
Nelineární SVM

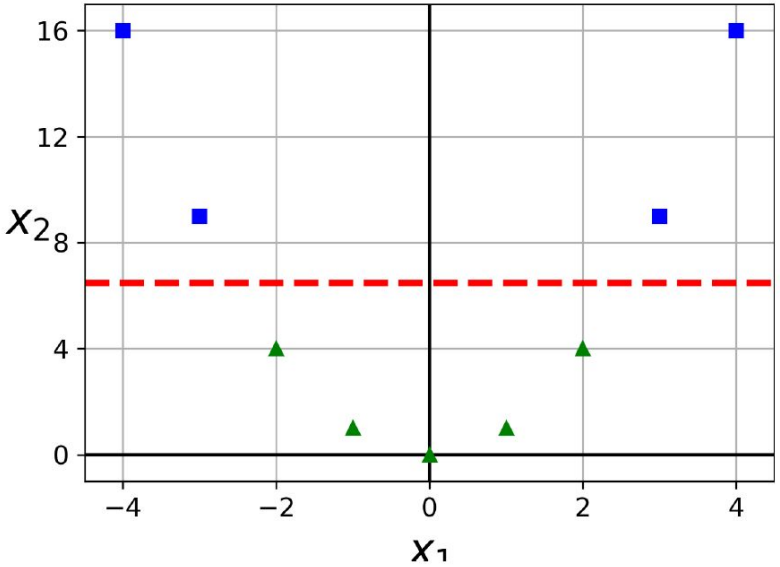
- mnoho datasetů má daleko k tomu, aby mohly být lineárně separovatelné
- je nutné použít nějaký trik a převést nelineární dataset na lineární
  - tzv. “kernel trick”

- dva základní způsoby:
  - polynomiální příznaky
  - podobnostní příznaky

Polynomiální příznaky pro nelineární SVM

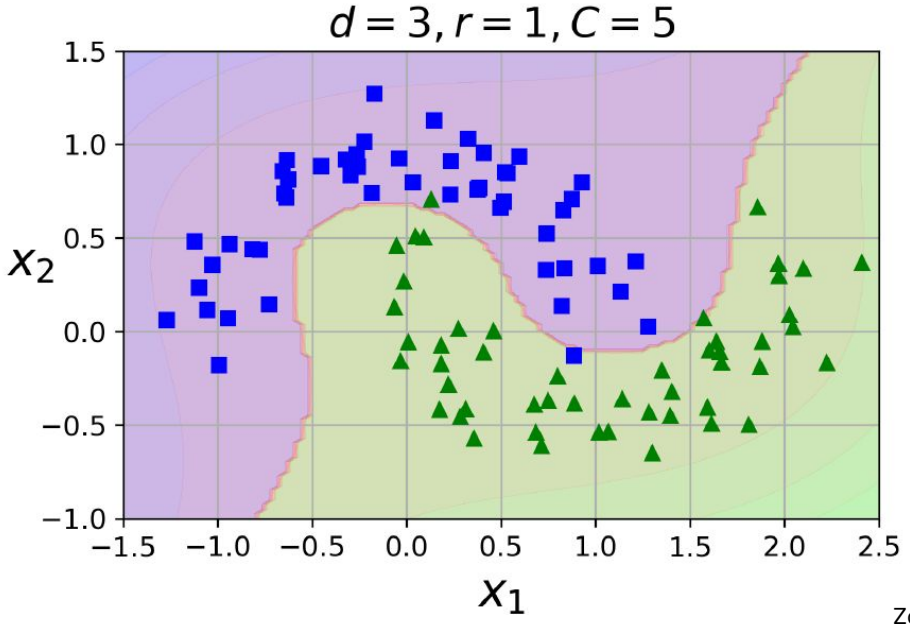
- vlevo máme dataset, který není lineárně separovatelný
- vpravo je dataset, kde je přidán příznak $x_2 = x_1^2$, toto již lineárně separovatelné je





Polynomiální příznaky pro nelineární SVM

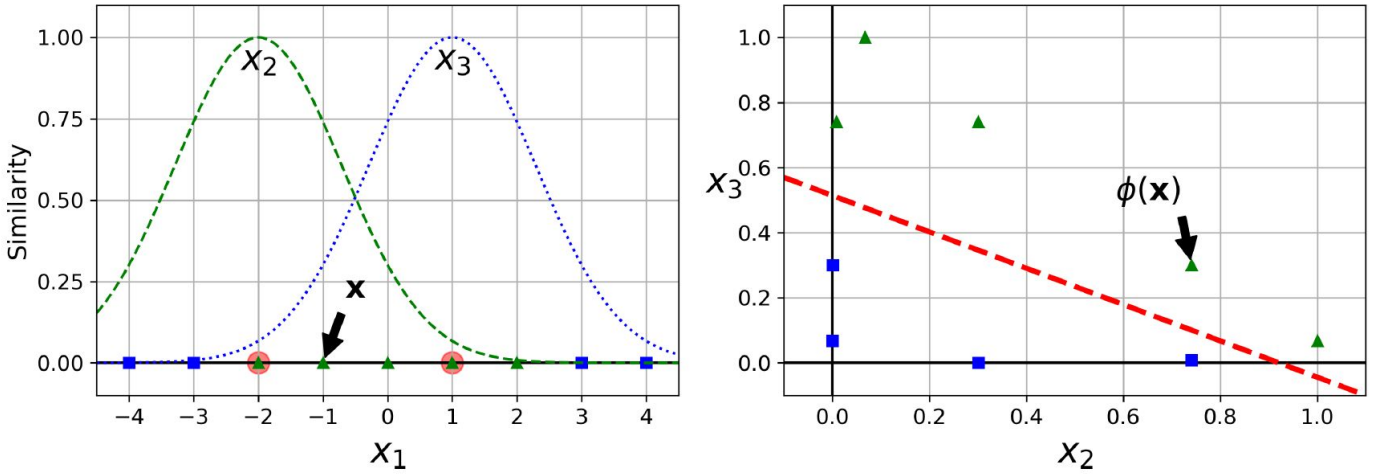
- SVC(kernel=”poly”, degree=3, C=5)
- ukázka na tzv. moons datasetu:



Zdroj: (Géron 2019)

Příznaky podobnosti pro nelineární SVM

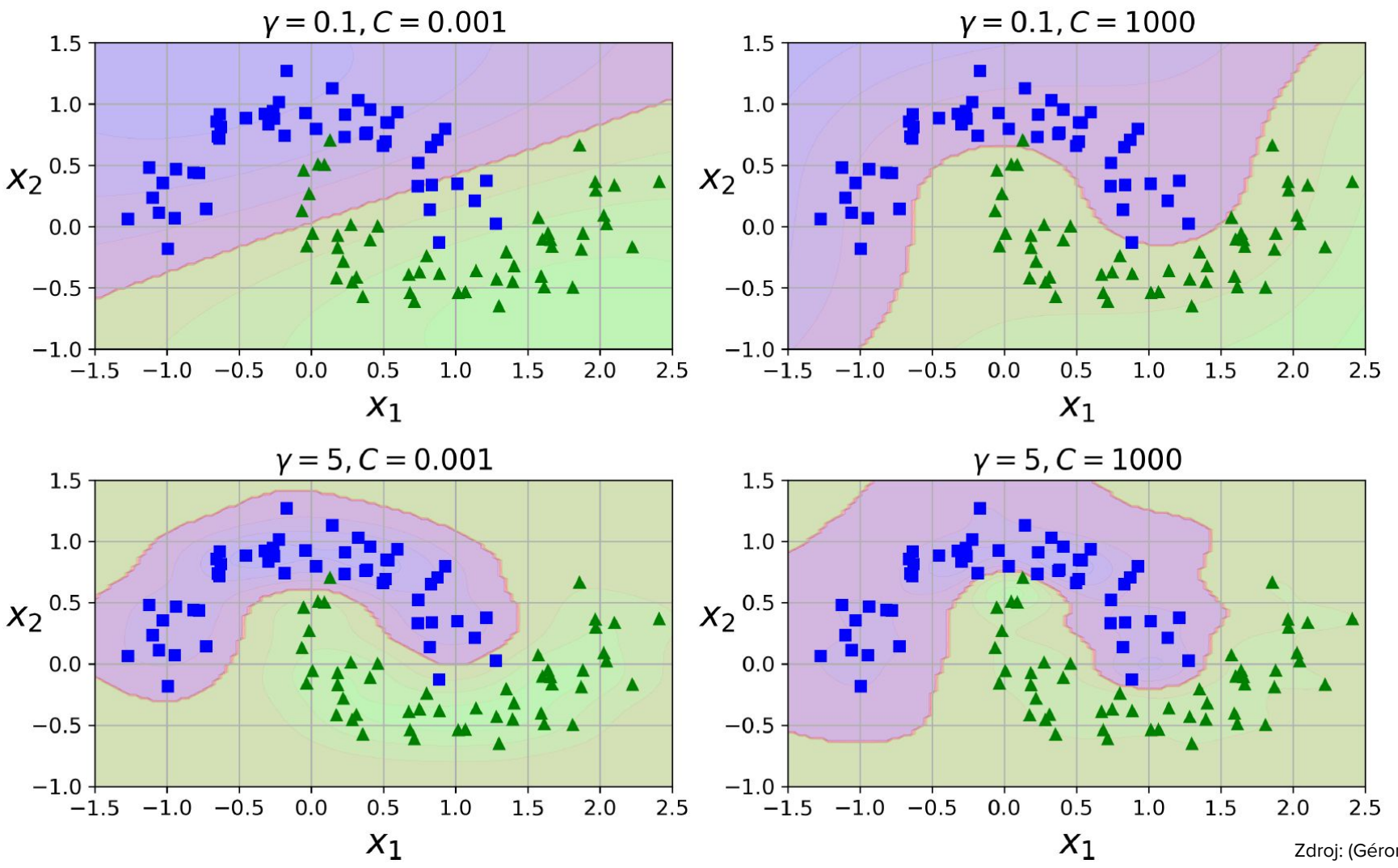
- jak moc jedna instance připomíná jinou (konkrétní) instanci v datasetu (příp. jiný pevný bod)
- jako míra podobnosti se použije RBF (Radial Basis Function)
  - křivka ve tvaru zvonu, vrací hodnoty 0 (velmi daleko) až 1 (přesná shoda)
  - vzniknou nové příznaky, které udávají, jak je daná instance daleko od zvolených bodů



Zdroj: (Géron 2019)

RBF kernel pro SVM

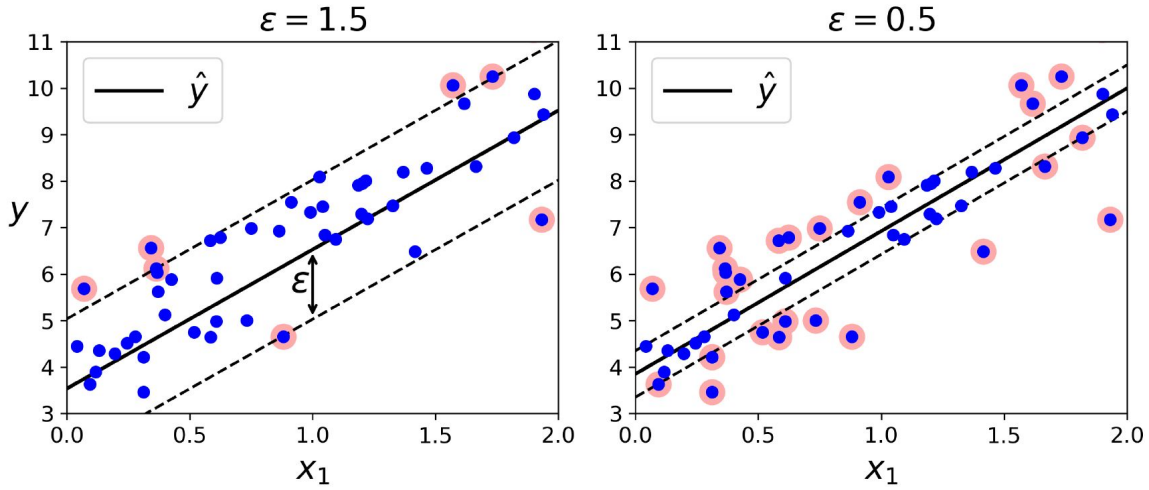
- abychom nemuseli volit, které body se mají použít pro porovnání podobnosti, zvolí se všechny instance z datasetu
  - pokud měl dataset m instancí a n příznaků, tak vznikne nový dataset o m instancích a m příznacích
- SVC(kernel=”rbf”, gamma=5, C=0.001)
  - gamma - čím vyšší, tím užší je tvar zvonu křivky RBF
    - vliv každé instance je pak menší
    - rozhodovací hranice je pak více nepravidelná
    - tj. gamma pracuje jako prvek regularizace (pokud se model přeučuje, snížit gamma; obdobné platí pro parametr C)



Zdroj: (Géron 2019)

SVM regrese

- metoda podpůrných vektorů jde použít i pro regresní problémy
- úloha je obrácená oproti klasifikaci:
  - snažíme se najít takovou “ulici”, která umožní na sebe vměstnat co nejvíce instancí
  - šířka ulice se nastavuje hyperparametrem epsilon



Zdroj: (Géron 2019)

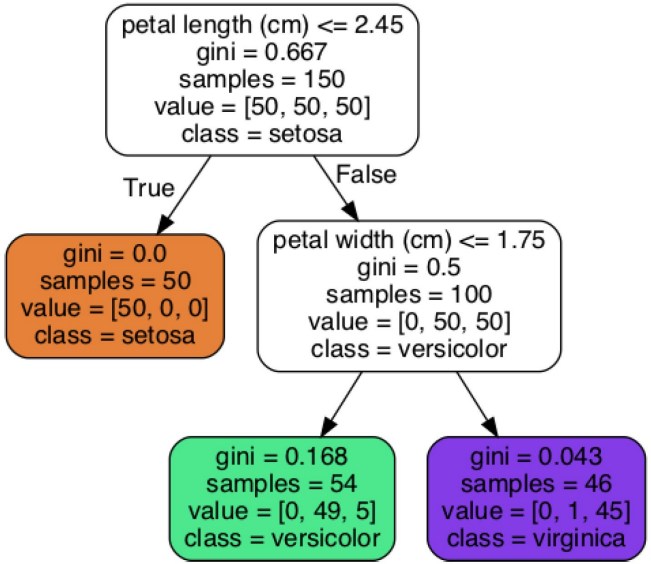
SVM regrese

- LinearSVR(epsilon=1.5)
- SVR(kernel=”poly”, degree=2, C=100, epsilon=0.1)

Rozhodovací stromy

Rozhodovací stromy

- univerzální a mocný model, který je schopen řešit klasifikační i regresní úlohy na velmi komplexních datasetech

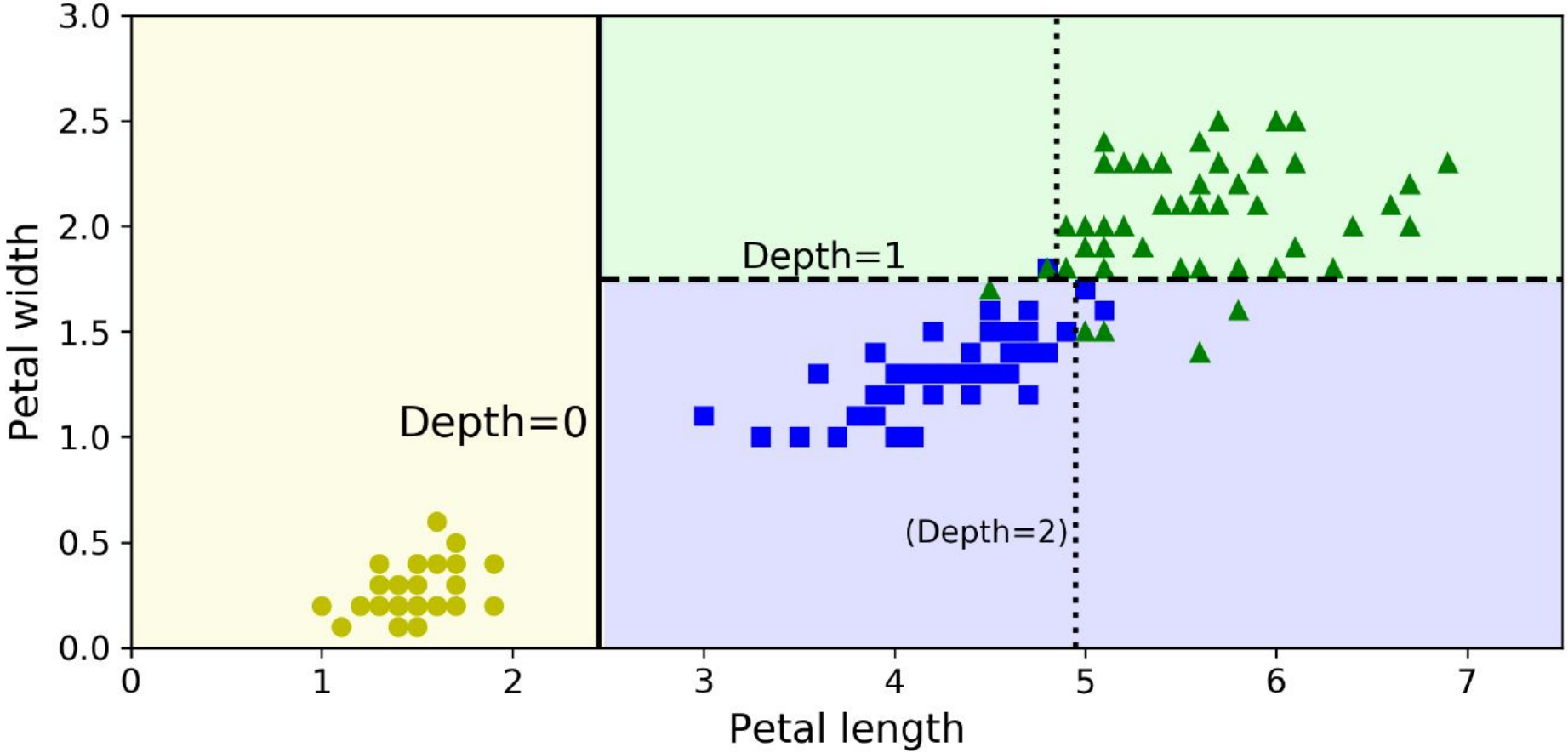


Zdroj: (Géron 2019)

Trénování a používání rozhodovacích stromů

- začneme v kořenovém uzlu a zodpovíme rozhodovací otázku
  - pokud je odpověď Ano, vydáme se vlevo, pokud Ne, vydáme se vpravo
- takto pokračujeme dokud nenarazíme na list (tj. nemá žádné další poduzly)
  - na listu vezmeme třídu, která je pro něj predikovaná, a to je celý výsledek

- atributy ve vizualizaci stromu
  - samples - na kolik instancí z trénovacího datasetu se tento list vztahuje
  - value - kolik instancí z jednotlivých tříd je na tomto listu
  - gini - měří “nečistotu” dat - pokud je uzel čistý (gini=0), tak všechny instance na tomto uzlu patří k jedné třídě



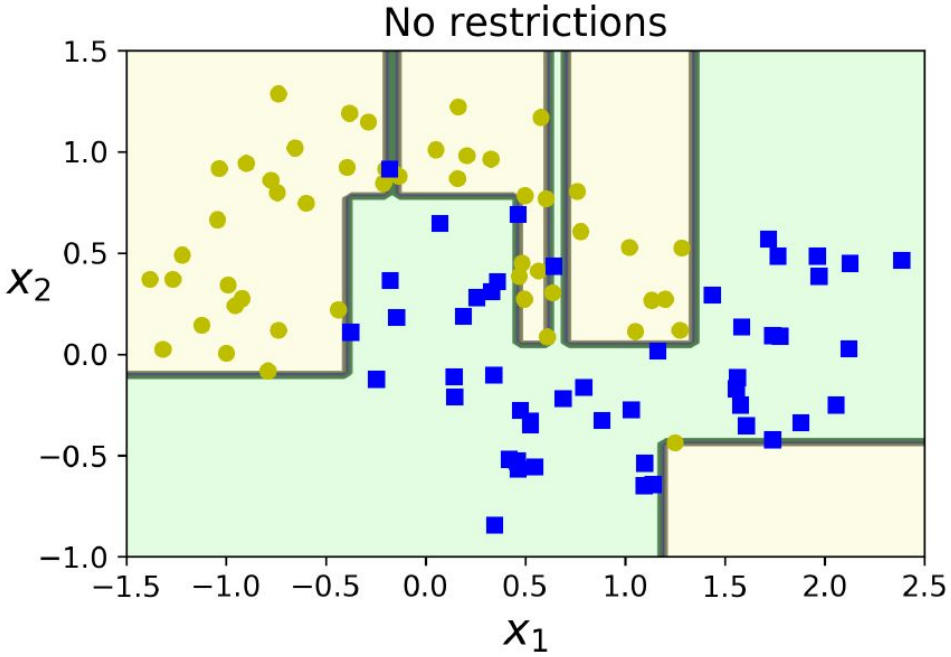
Zdroj: (Géron 2019)

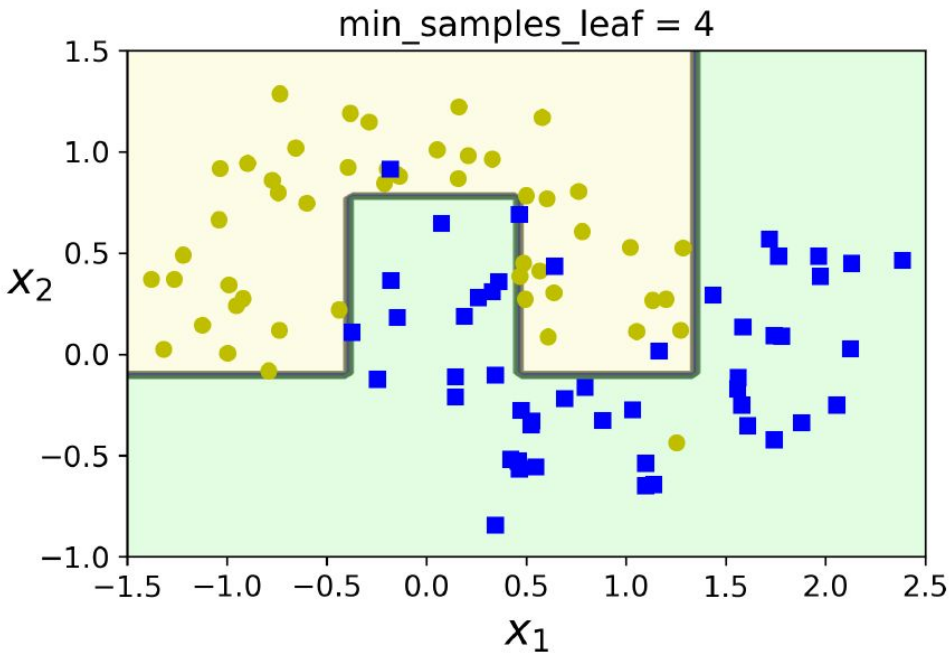
Vlastnosti rozhodovacích stromů

- rozhodovací stromy nevyžadují téměř žádnou přípravu dat
  - není nutné škálování apod.
- white box model - rozhodovací stromy jsou intuitivní a jejich rozhodovací postup je jednoduchý na interpretaci
  - oproti tomu neuronové sítě nebo i náhodné lesy jsou black box modely, do kterých není dobře vidět
- bez regularizace velmi náchylné na přeučení
- dokáží (na rozdíl od např. SVM) určit pravděpodobnost, že instance patří k určité třídě (zde řešeno jako poměrové zastoupení jednotlivých tříd v posledním uzlu - listu)
- rozhodovací hranice jsou ortogonální (kolmé na osy, viz dále)

Trénovací algoritmus

- algoritmus v každém kroku rozdělí dataset podle jednoho příznaku *k* a jeho prahové hodnoty t_k (“petal length < 2.45 cm”)
  - dvojice *k* a t_k se hledá tak, aby po jejich použití vznikly co nejčistší podmnožiny
  - nákladová funkce $J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$
    - G - nečistota (impurity) v levé a pravé podmnožině
    - $m_{\text{left/right}}$ - počet instancí v levé a pravé podmnožině
- parametry (zejména s ohledem na regularizaci modelu)
  - max_depth - maximální počet úrovní, při jejich dosažení algoritmus skončí
  - min_samples_split - minimální počet instancí, které musí na uzlu být, aby mohlo dojít k jeho dalšímu rozdělení
  - min_samples_leaf - minimální počet instancí na listu

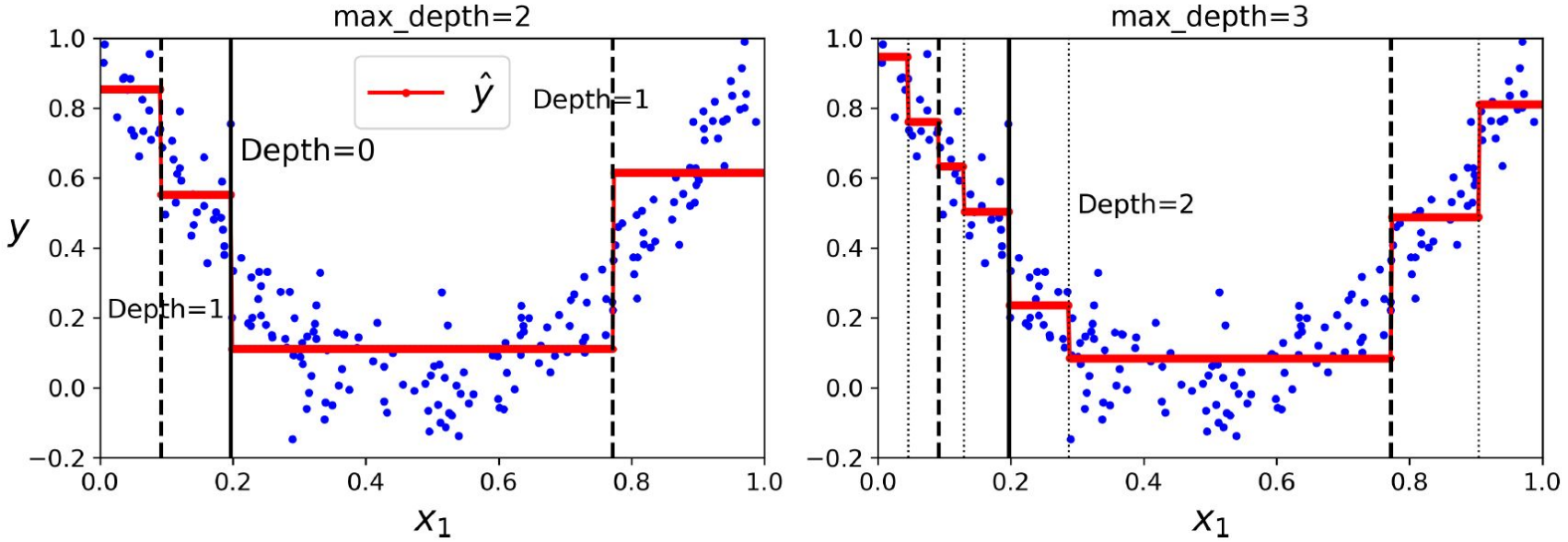




Zdroj: (Géron 2019)

Regrese s rozhodovacími stromy

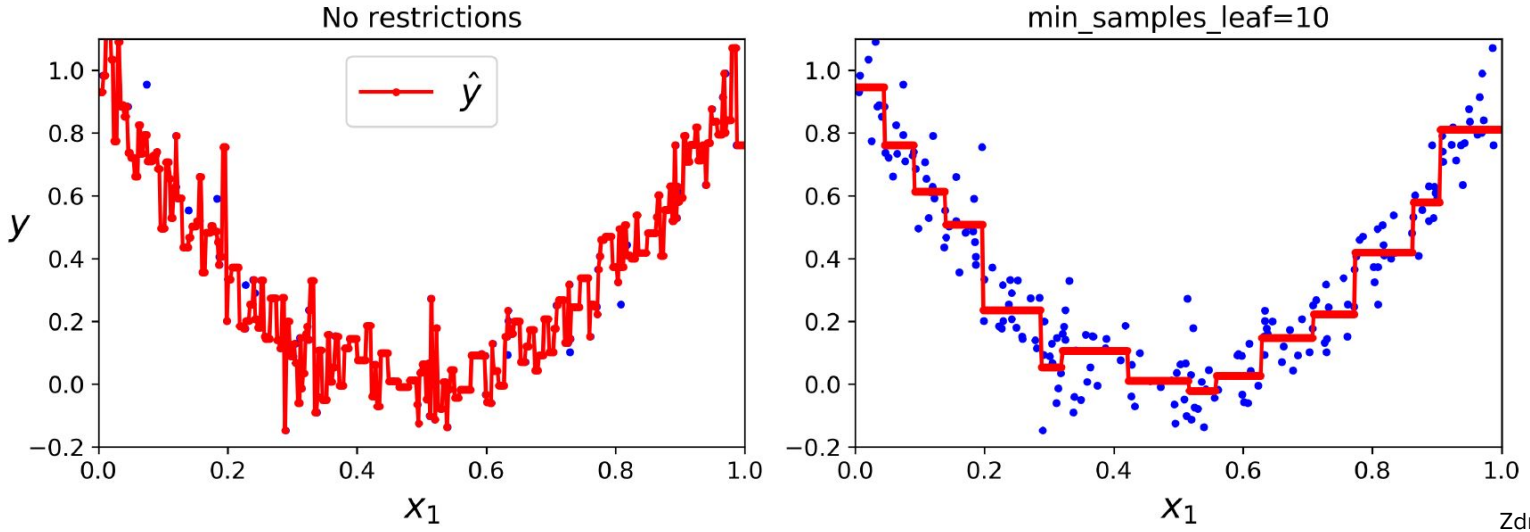
- místo predikce třídy je v každém listu predikce hodnoty
  - hodnota v listu je průměrná hodnota z trénovacích instancí, které na tento list patří



Zdroj: (Géron 2019)

Regrese s rozhodovacími stromy

- místo gini impurity je při regresi dataset dělen tak, aby se minimalizovala chyba MSE
- i v případě regrese jsou rozhodovací stromy náchylné na přeučení



Zdroj: (Géron 2019)

Ensemble Learning

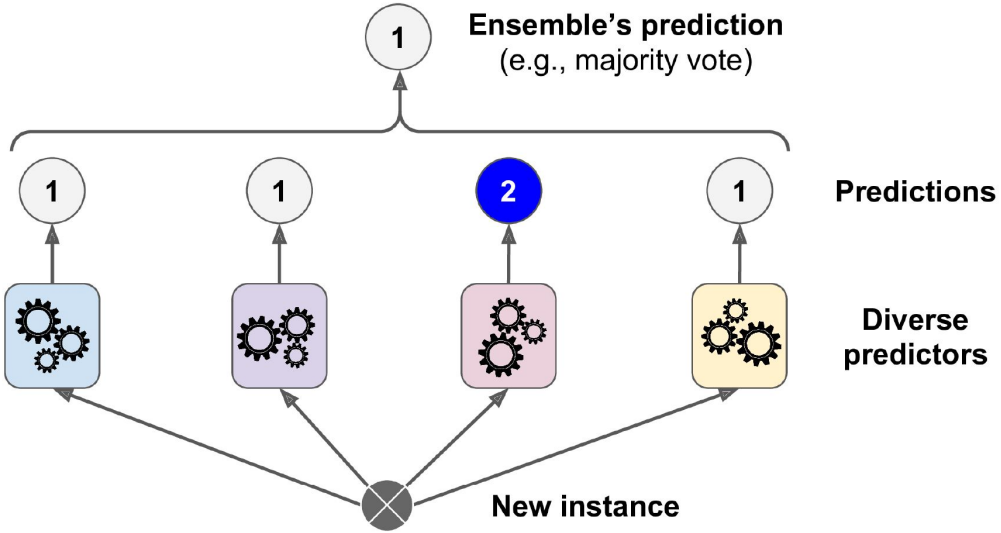
Ensemble Learning

- založeno na předpokladu, že “víc hlav víc ví” (v tomto případě víc prediktorů)
- pokud vezmeme agregovanou predikci skupiny různých prediktorů, získáme většinou lepší výsledek, než by byl nejlepší individuální výsledek jednoho prediktoru
- skupina prediktorů = ensemble => Ensemble Learning

- např. Random Forest - natrénujeme skupinu rozhodovacích stromů, každý na jiné podmnožině datasetu a jako výsledek vezmeme predikci, která má nejvíce hlasů (tj. vyskytla se nejčastěji)
  - jednoduchá a přitom jedna z nejúčinnějších metod
- obvykle využito na konci projektu, když máme několik dobrých modelů, tak je zkombinujeme pomocí ensemble metod a získáme ještě o něco lepší výsledný model
  - vítězné modely v Kaggle i jiných soutěžích často využívají ensemble metody
  - např. v Titanic datasetu z minulé hodiny se dalo dosáhnout správnosti přes 83,8 %

Voting Classifiers

- hlasovací klasifikátor vezme výstupy dílčích klasifikátorů a jako svůj výstup zvolí ten, který je nejčastější
  - viz ukázka na Titanic datasetu (kombinace SVM, Random Forest, XGBoost, Logistická regrese)



Zdroj: (Géron 2019)

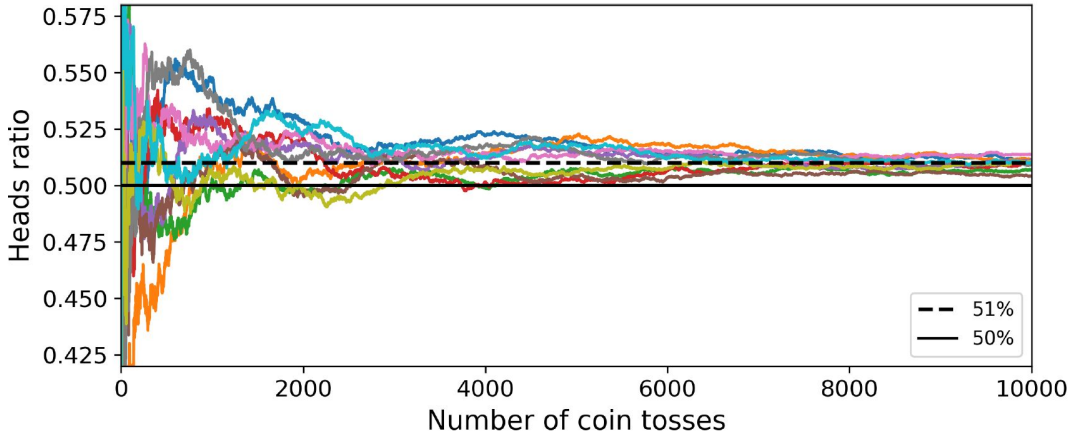
Voting Classifiers

- hard voting
  - berou se v potaz jen predikce tříd
- soft voting
  - bere se v potaz pravděpodobnost třídy
    - klasifikátory, které si jsou pro danou instanci hodně jisté svou predikcí, mají větší váhu
  - jednotlivé klasifikátory musí podporovat metodu predict_proba
  - obvykle dosahuje lepších výsledků

- ačkoliv to nemusí být na první pohled intuitivní, hlasovací klasifikátor většinou dosahuje lepších výsledků než nejlepší model v něm obsažený

Princip Voting Classifiers

- mějme 1000 klasifikátorů, kdy každý má správnost pouze 51 %
  - tedy sotva o něco lepší výsledek než zcela náhodný klasifikátor (na vyváženém datasetu)
- pokud jako výsledek vezmeme jejich společnou nejčastější predikci, dosáhneme celkové správnosti 75 %
  - zákon velkých čísel (za předpokladu, že jednotlivé klasifikátory jsou nezávislé)



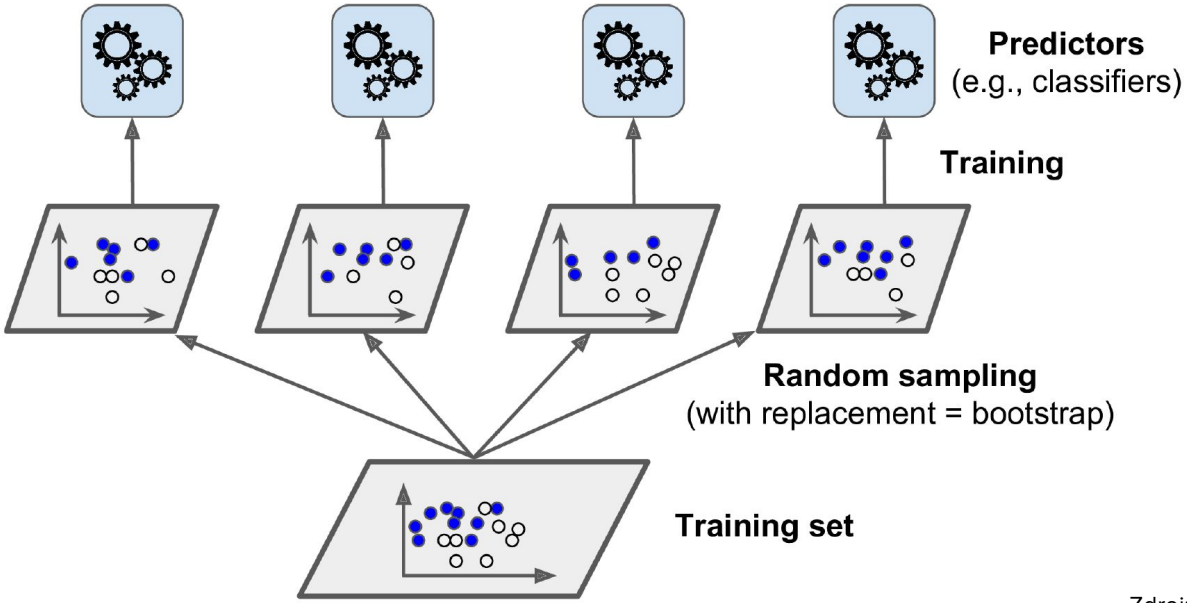
Zdroj: (Géron 2019)

Princip Voting Classifiers

- nezávislost klasifikátorů lze zvýšit tak, že použijeme co nejrůznější algoritmy strojového učení
  - toto zvyšuje šanci, že i chyby, které budou klasifikátory dělat, budou velmi odlišné

Bagging

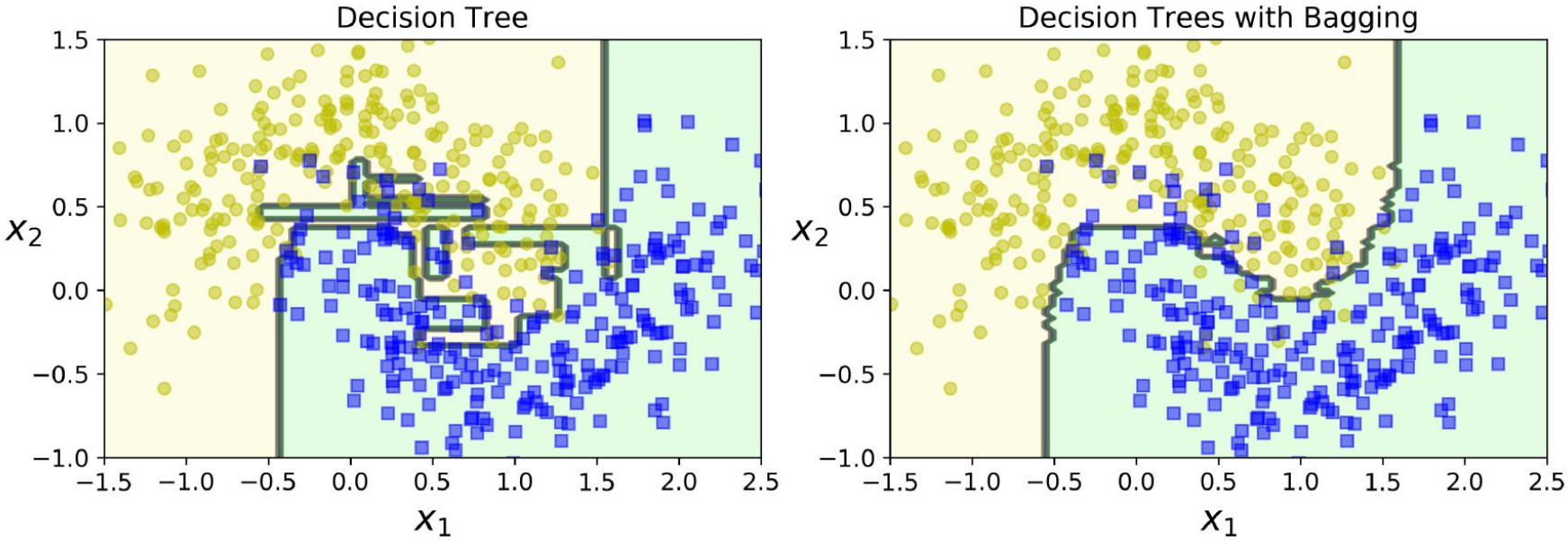
- druhou možností jak získat rozdílné klasifikátory je použít stejný model, ale trénovat ho na různých podmnožinách datasetu (z hlediska instancí i příznaků)



Zdroj: (Géron 2019)

Bagging s rozhodovacími stromy

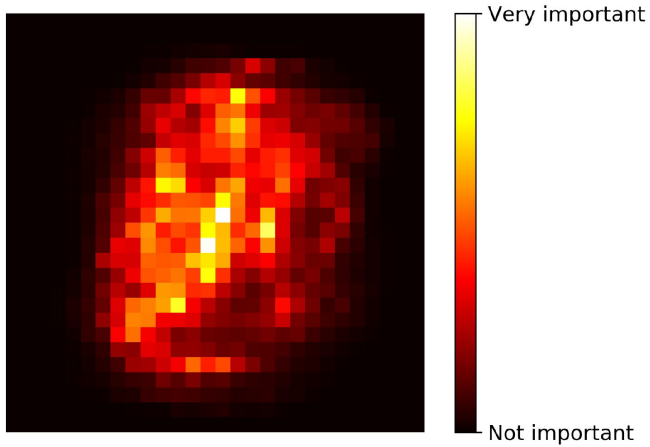
= Random Forest (náhodné lesy)



Zdroj: (Géron 2019)

Důležitost jednotlivých příznaků u Random Forest

- velkou výhodou náhodných lesů je, že dokáží spočítat relativní významnost jednotlivých příznaků
  - jak moc jednotlivé uzly, které používají daný příznak, dokážou snížit gini impurity
  - viz Titanic dataset



Zdroj: (Géron 2019)

Boosting

- na rozdíl od předchozích ensemble metod je boosting založen na tom, že jednotlivé prediktory na sebe postupně navazují
  - tedy nejsou vedle sebe, ale v řadě za sebou
  - cílem každého navazujícího prediktoru je opravovat chyby toho předchozího

- dva populární přístupy:
  - AdaBoost
  - Gradient Boosting
    - na ten se podíváme podrobněji

Gradient Boosting

- pracuje na principu, že se postupně za prediktor přidává další prediktor, který opravuje chybu toho předchozího
  - opravou chyby se myslí natrénování dalšího prediktoru na tzv. residual error

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)
```

...

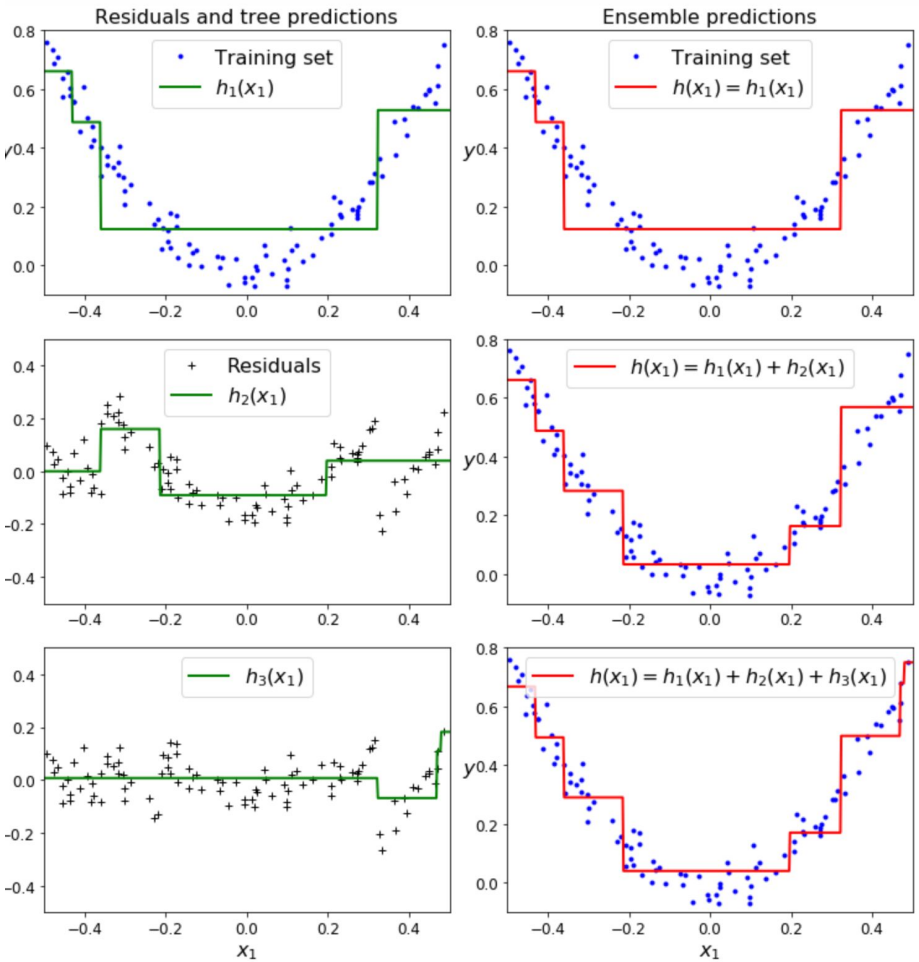
```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)
```

...

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```

...

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```



Zdroj: (Géron 2019)

GradientBoosting Regressor/Classifier

- v sklearn jsou k dispozici třídy, které pohodlně poskytují výše uvedené
  - GradientBoostingClassifier
  - GradientBoostingRegressor

- optimalizovaná implementace Gradient Boosting je k dispozici v XGBoost
  - Extreme Gradient Boosting
  - rychlé a škálovatelné
  - v současnosti je XGBoost často součástí vítězných řešení v ML soutěžích

Zdroje

- Coelho, L. P.; Richert, W. (2013) Building machine learning systems with Python. Birmingham: Packt Publishing. ISBN 978-1-78216-140-0.

- Géron, A. (2019) Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. O'Reilly Media, Inc. ISBN 9781492032649.

- Chollet, F. (2019) Deep Learning v jazyku Python. Knihovny Keras, TensorFlow. Grada Publishing, a.s. ISBN 978-80-247-3100-1.

- Segaran, T. (2007) Programming collective intelligence: building smart web 2.0 applications. Beijing: O'Reilly Media. ISBN 0-596-52932-5.