ML Drzewa i Lasy

ML Drzewa i Lasy

Agenda

1. **Bias-variance tradeoff**
2. Overfitting, underfitting
3. Cross-validation
4. Struktura algorytmu drzewa decyzyjnego i lasów losowych
5. Parametry uczenia drzewa decyzyjnego i lasów losowych

ML Drzewa i Lasy

Agenda

1. Bias-variance tradeoff
2. **Overfitting, underfitting**
3. Cross-validation
4. Struktura algorytmu drzewa decyzyjnego i lasów losowych
5. Parametry uczenia drzewa decyzyjnego i lasów losowych

ML Drzewa i Lasy

Agenda

1. Bias-variance tradeoff
2. Overfitting, underfitting
3. **Cross-validation**
4. Struktura algorytmu drzewa decyzyjnego i lasów losowych
5. Parametry uczenia drzewa decyzyjnego i lasów losowych

ML Drzewa i Lasy

Agenda

1. Bias-variance tradeoff
2. Overfitting, underfitting
3. Cross-validation
4. **Struktura algorytmu drzewa decyzyjnego i lasów losowych**
5. Parametry uczenia drzewa decyzyjnego i lasów losowych

ML Drzewa i Lasy

Agenda

1. Bias-variance tradeoff
2. Overfitting, underfitting
3. Cross-validation
4. Struktura algorytmu drzewa decyzyjnego i lasów losowych
5. **Parametry uczenia drzewa decyzyjnego i lasów losowych**

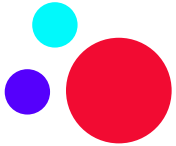
ML Drzewa i Lasy

Model - definicja

Model jest odwzorowaniem X→Y otrzymanym na podstawie danych. Jest to estymator $f(x)$ odwzorowania $f(x)$ dla, którego zachodzi

$$Y = f\ (X)\ +\ \varepsilon$$

X – predykatory (wejście, zmienne objaśniające, zmienne niezależne),
Y – odpowiedź (wyjście, zmienna objaśniana, zmienna zależna),
ε – oznacza błąd losowy.



ML Drzewa i Lasy

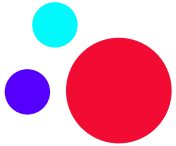
Obciążenie modelu - bias

Obciążenie modelu oznacza różnicę (błąd) między wartością oczekiwaną predykcji $E\,f(x)$, a prawdziwą (nieznaną) wartością funkcji $f(x)$.

$$Bias\ f(x) = E\,[f(x)] - f(x)$$

W praktyce obciążenie modelu oznacza, że model zawsze dokonuje predykcji z nadmiarem (niedomiarem).

Obciążenie powinno być jak najmniejsze.



ML Drzewa i Lasy

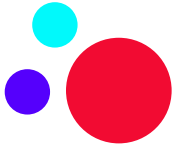
Wariancja modelu - variance

Wariancja modelu opisuje dyspersję predykcji.

$$Var\left(f(x)\right) = E\left[\left(f(x) - E[f(x)]\right)^2\right]$$

Duża wariancja oznacza, że predykcje dla nieodległych od siebie wartości x będą mocno zróżnicowane, "rozstrzelone".

Wariancja powinna być jak najmniejsza.



ML Drzewa i Lasy

Błąd, wariancja, obciążenie

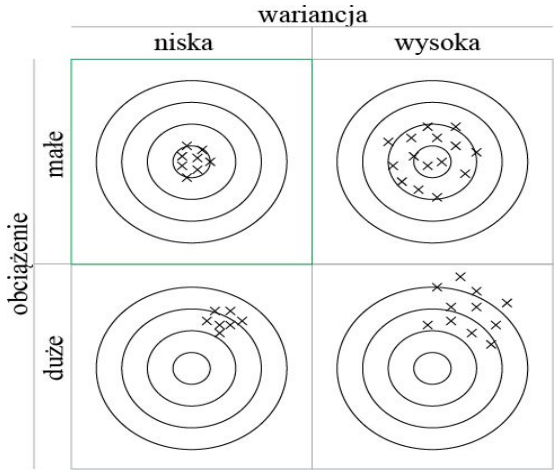
$$E(MSE) = Var\big(f(x)\big) + \big[Bias\big(f(x)\big)\big]^2 + Var(\varepsilon)$$

$$\begin{pmatrix} Wartość \\ oczekiwana \\ błędu \\ kwadratowego \end{pmatrix} = \begin{pmatrix} Wariancja \\ modelu \end{pmatrix} + \begin{pmatrix} Obciążenie \\ modelu \end{pmatrix}^2 + \begin{pmatrix} Wariancja \\ błędu \\ losowego \end{pmatrix}$$

ML Drzewa i Lasy

bias-variance tradeoff

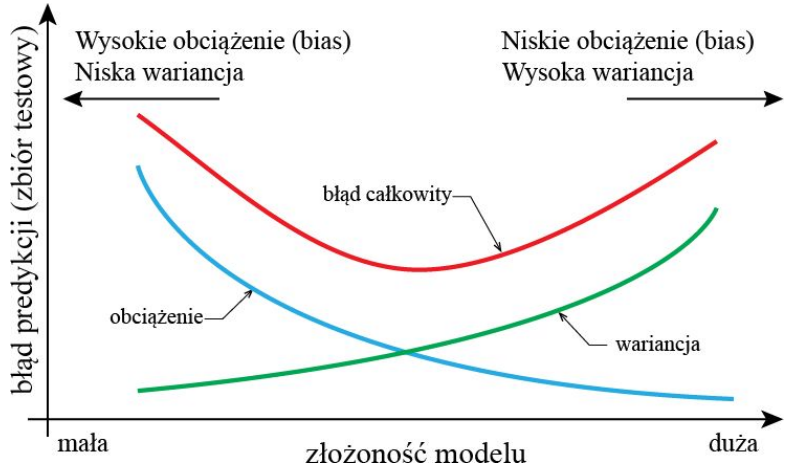
Czyli z czym i o co walczymy?



ML Drzewa i Lasy

bias-variance tradeoff

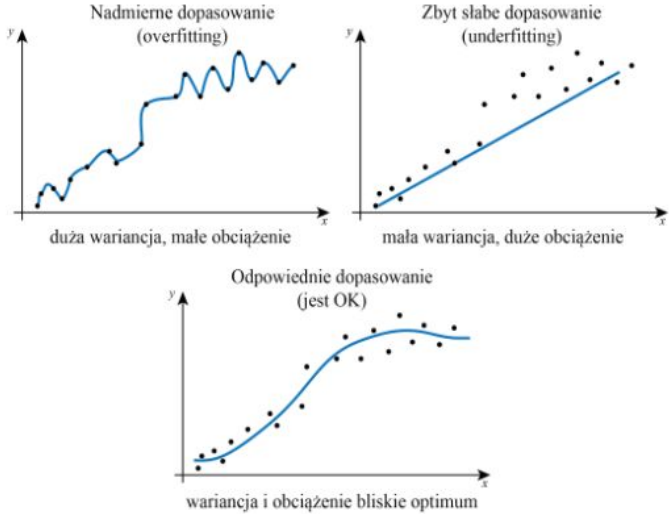
Błąd predykcji.

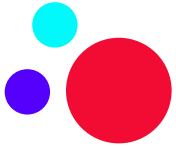


ML Drzewa i Lasy

overfitting/underfitting

Wygląd predykcji.





ML Drzewa i Lasy

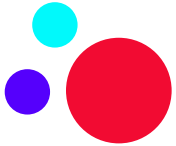
overfitting

Cechy charakterystyczne:

1. Doskonałe dopasowanie do danych treningowych
2. Słaba generalizacja
3. Wzrost błędu na danych testowych

Przyczyny overfittingu:

1. Zbyt skomplikowany model
2. Niewystarczająco duże dane treningowe
3. Brak regularyzacji



ML Drzewa i Lasy

underfitting

Cechy charakterystyczne:

1. Słabe dopasowanie do danych treningowych
2. Wysoki błąd na danych treningowych
3. Podobny błąd na danych testowych

Przyczyny underfittingu:

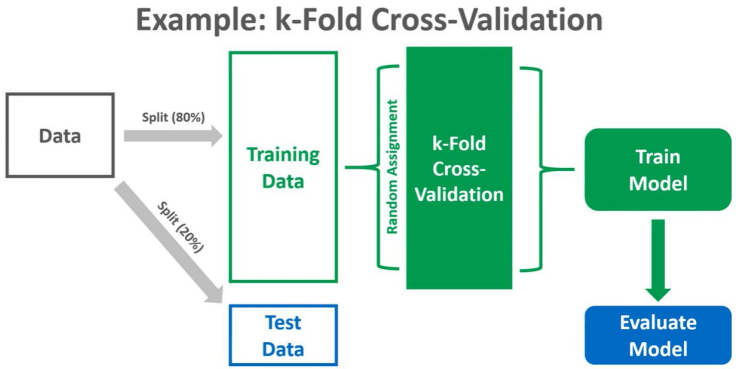
1. Zbyt prosty model
2. Brak danych treningowych
3. Brak dostosowania parametrów

ML Drzewa i Lasy

cross-validation

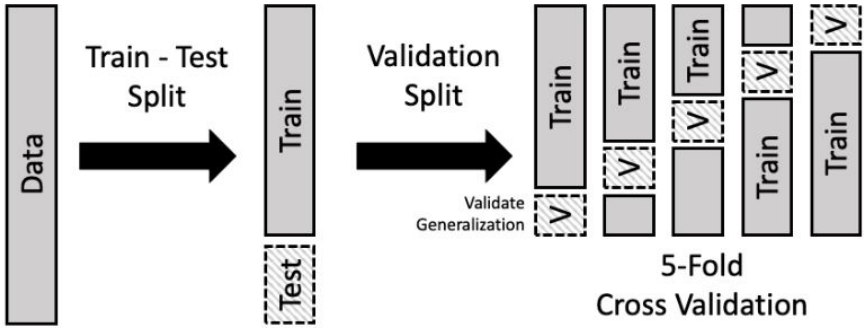
Walidacja krzyżowa (ang. cross-validation, CV) polega na wielokrotnym trenowaniu modelu na wybranym podzbiorze zbioru treningowego i testowaniu na pozostałej części zbioru treningowego.

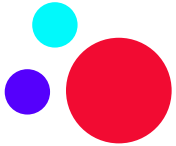
Jest to sposób trenowania modelu minimalizujący wariancję modelu oraz jego obciążenie.



ML Drzewa i Lasy

Proces cross-validation





ML Drzewa i Lasy

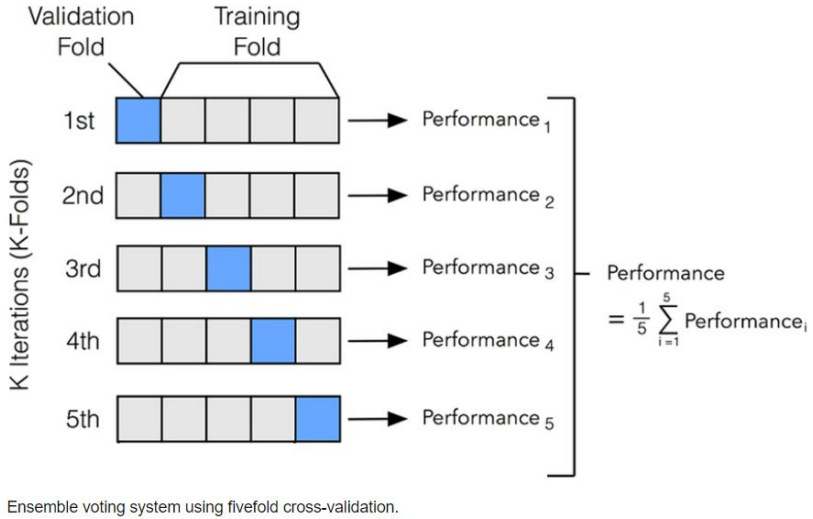
cross-validation - korzyści

1. Lepsza ocena generalizacji
2. Unikanie wpływu losowego podziału danych
3. Lepsze wykrywanie overfittingu

ML Drzewa i Lasy

k-fold cross-validation

k-fold cross validation oznacza walidację krzyżową przy podziale zbioru na k podzbiorów, a następnie k-krotne trenowanie.

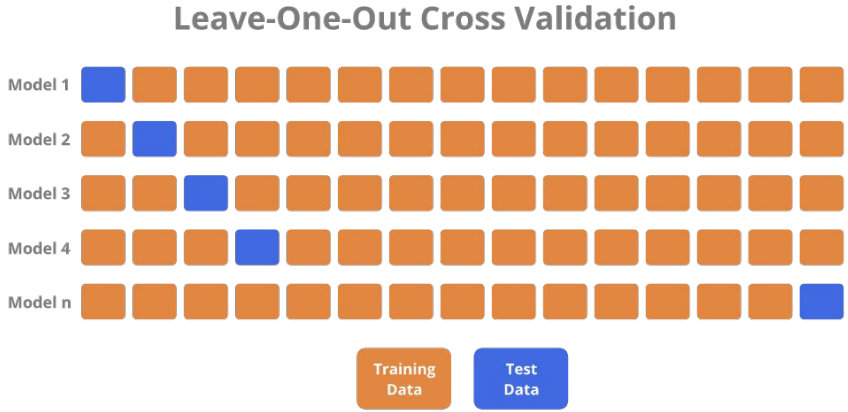


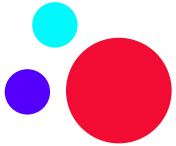
Ensemble voting system using fivefold cross-validation.

ML Drzewa i Lasy

cross-validation

Leave One Out (LOO CV) oznacza walidację krzyżową przy k=n, gdzie n jest równe liczbie elementów zbioru testowego.

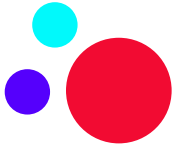




ML Drzewa i Lasy

k-fold cross-validation - (implementacja sklearn)

```
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
import numpy as np

data = load_iris()

X, y = data.data, data.target
```



ML Drzewa i Lasy

k-fold cross-validation - (implementacja sklearn)

num_folds = 5

kf = **KFold**(n_splits=num_folds, shuffle=True, random_state=42)

model = LogisticRegression()

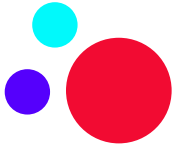
ML Drzewa i Lasy

k-fold cross-validation - (implementacja sklearn)

```
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model.fit(X_train, y_train)

    predictions = model.predict(X_test)

    accuracy = accuracy_score(y_test, predictions)
```

5 - folds

Accuracy: 1.0
Accuracy: 1.0
Accuracy: 0.9333333333333333
Accuracy: 0.9666666666666667
Accuracy: 0.9666666666666667

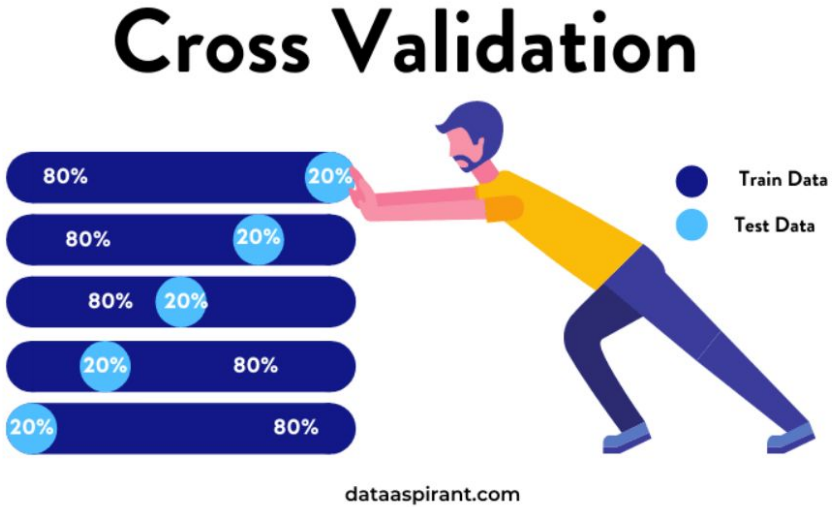


Zadanie 13.1 (instrukcja)

Zastosuj metodę LeaveOneOut cross validation do zbioru Iris, analogicznie do przeprowadzonej analizy podczas lekcji używając do tego pętli.

ML Drzewa i Lasy

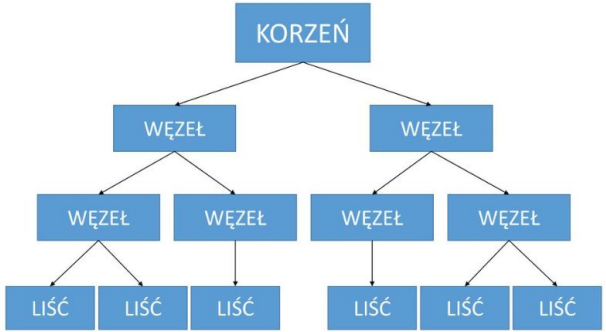
Podsumowanie



ML Drzewa i Lasy

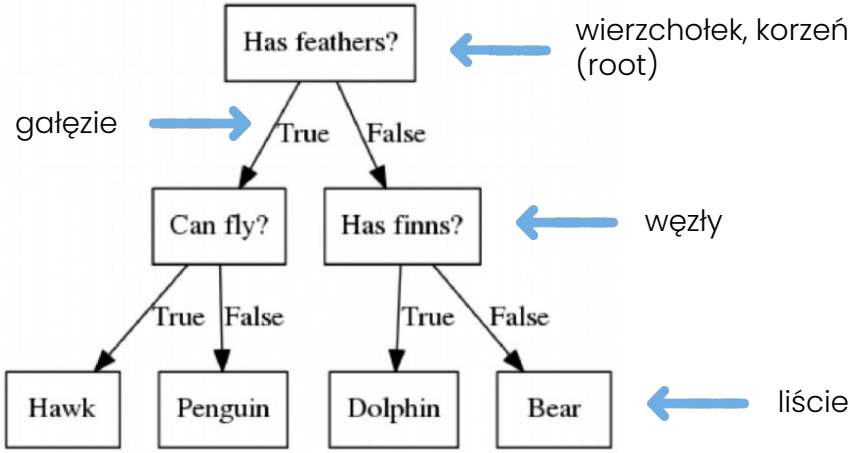
Drzewo decyzyjne

- Metoda wspomagania procesów decyzji.
- Model używany do zadań regresji i klasyfikacji.
- Intuicyjny model opierający się na podziale danych przez serię porównań.



ML Drzewa i Lasy

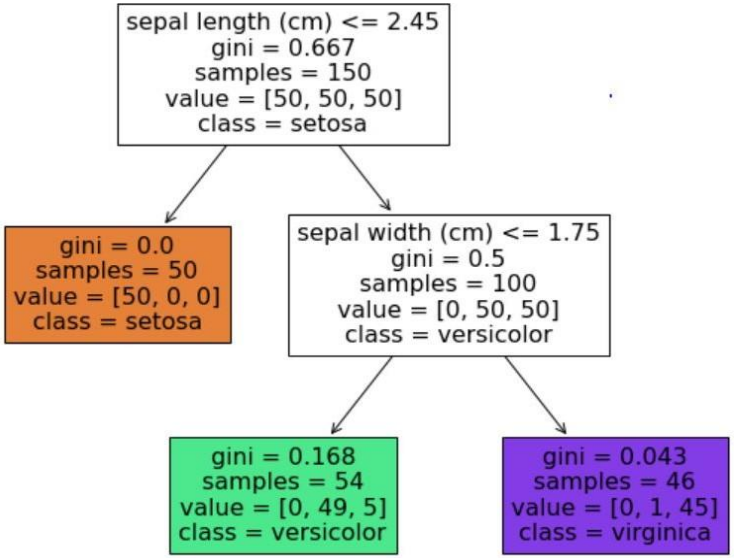
Wizualizacja algorytmu

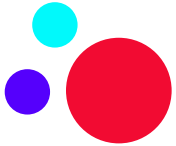


ML Drzewa i Lasy

Algorytm CART

Classification and Regression Trees:





ML Drzewa i Lasy

Kryteria podziału

Podział polega na jak najlepszym rozdzieleniu podgrupy na części - tak aby w węzłach dzieci różnorodność była jak najmniejsza.

Miara różnorodności:

- 0 - wszystkie obserwacje należą do tej samej klasy,
- wartość maksymalna - rozkład przynależności do klas jest jednostajny.

ML Drzewa i Lasy

Kryteria nieczystości

Indeks Giniego:

$$I_G = 1 - \sum_{j=1}^{c} p_j^2$$

p_j: część próbek należąca do klasy c dla danego węzła

ML Drzewa i Lasy

Kryteria nieczystości

Entropia:

$$I_H = -\sum_{j=1}^{c} p_j log_2(p_j)$$

p_j: część próbek należąca do klasy c dla danego węzła.

*To jest definicja entropii dla wszystkich niepustych klas ($p \neq 0$).

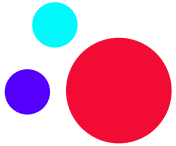
Entropia wynosi 0, jeśli wszystkie próbki w węźle należą do tej samej klasy.

ML Drzewa i Lasy

Funkcja kosztu dla algorytmu CART

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

where $\begin{cases} G_{\text{left/right}} \text{ measures the impurity of the left/right subset,} \\ m_{\text{left/right}} \text{ is the number of instances in the left/right subset.} \end{cases}$

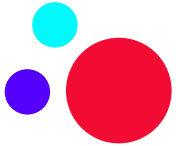


ML Drzewa i Lasy

Funkcja kosztu dla algorytmu CART

Information Gain:

różnica miary nieczystości rodzica i funkcji J, mówi o przyroście informacji po dodaniu kolejnego węzła. Pozwala na dokonanie doboru warunków kolejnych podziałów.



ML Drzewa i Lasy

Moment stopu

Gdy osiągniemy maksymalną głębokość drzewa (max_depth) albo nie można znaleźć podziału, który zlikwiduje nieczystość (impurity).

ML Drzewa i Lasy

Regresja przy użyciu drzew decyzyjnych

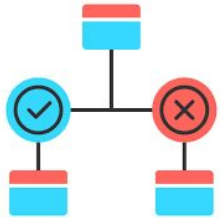
Drzewa decyzyjne możemy również stosować dla problemów regresji. Wówczas jako kryterium stosujemy zwykle MSE (Mean square error).

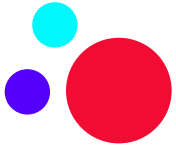
Funkcja kosztu:

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

ML Drzewa i Lasy

Problemy drzew decyzyjnych



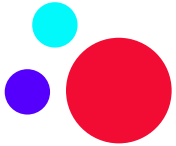


ML Drzewa i Lasy

overfitting

Drzewa dążą do tego, by rozrastać się aż do uzyskania czystych podzbiorów w liściach. Często wiąże się to z tym, że w praktyce takie drzewo “zapamiętuje” zbiór treningowy.

Aby zredukować efekty overfittingu możemy manipulować parametrami modelu.



ML Drzewa i Lasy

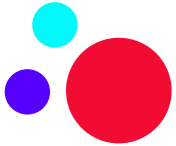
API sklearn.tree

sklearn.tree.DecisionTreeClassifier

class `sklearn.tree.` **`DecisionTreeClassifier`**(**, criterion='gini', splitter='best', max_depth=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_decrease=0.0, min_impurity_split=None, class_weight=None, ccp_alpha=0.0*) [source]

sklearn.tree.DecisionTreeRegressor

class `sklearn.tree.` **`DecisionTreeRegressor`**(**, criterion='mse', splitter='best', max_depth=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_decrease=0.0, min_impurity_split=None, ccp_alpha=0.0*) [source]



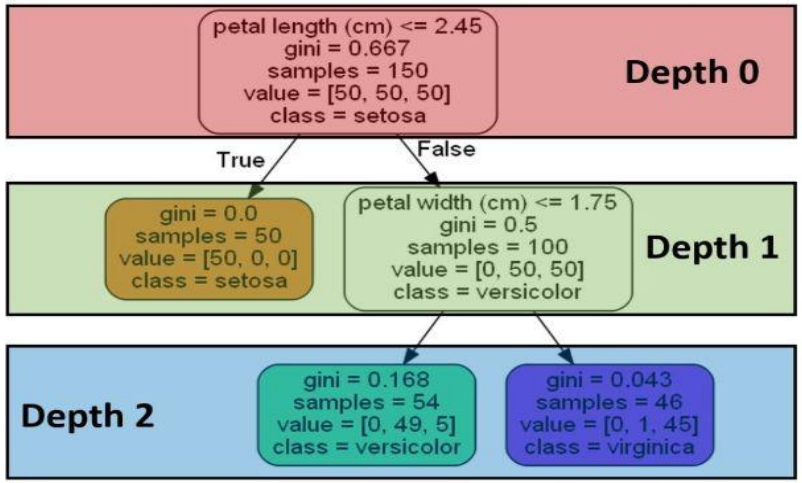
ML Drzewa i Lasy

Parametry drzewa decyzyjnego

- **max_depth -** głębokość drzewa
- **min_samples_leaf -** minimalna liczba obserwacji w liściu
- **max_leaf_nodes -** maksymalna liczba liści w drzewie
- **max_features -** maksymalna liczba zmiennych rozważanych w podziale
- **ccp_alpha -** jak mocno przycinane są drzewa
- **criterion -** kryterium nieczystości `gini`, `entropy`
- **random_state -** algorytm stochastyczny (!)
- **class_weight -** ważenie klas przy niezbalansowaniu

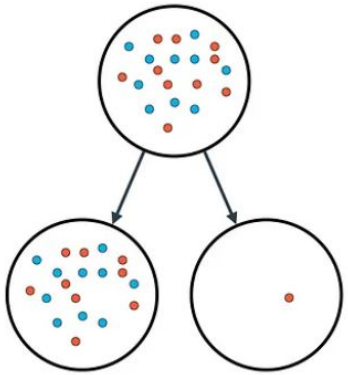
ML Drzewa i Lasy

max_depth

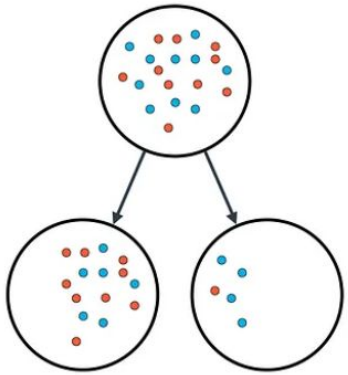


ML Drzewa i Lasy

min_samples_leaf



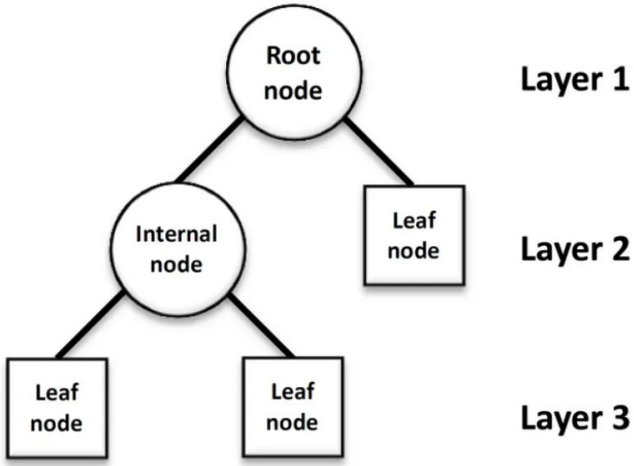
Minimum samples per leaf = 1



Minimum samples per leaf = 5

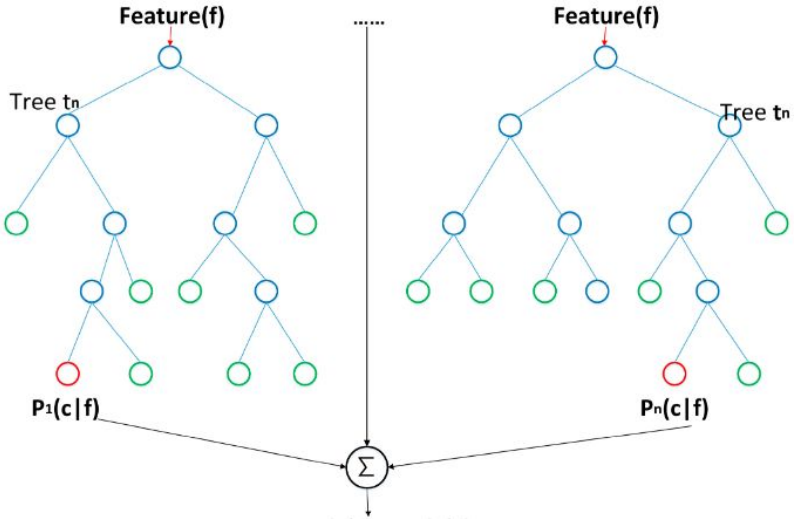
ML Drzewa i Lasy

max_leaf_nodes



ML Drzewa i Lasy

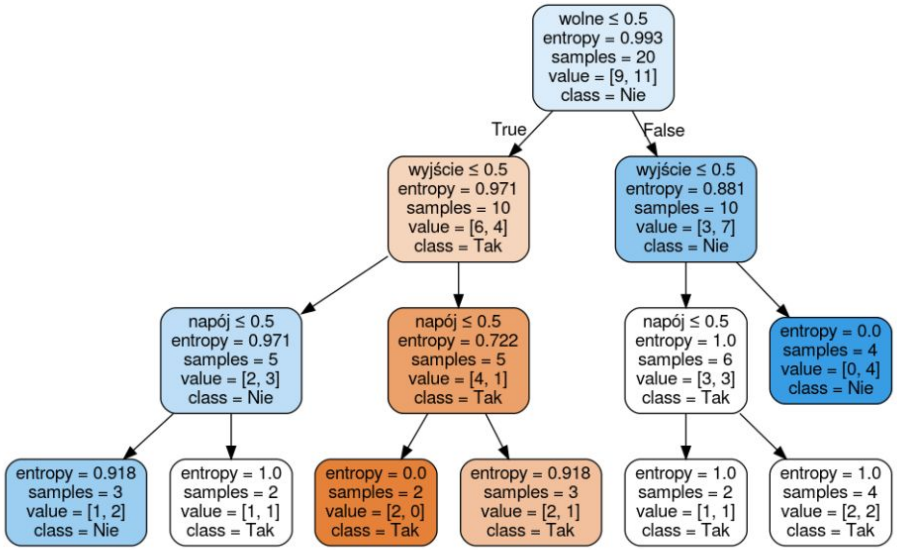
max_features



$$P(c|f) = \sum_{1}^{n} P_n(c|f)$$

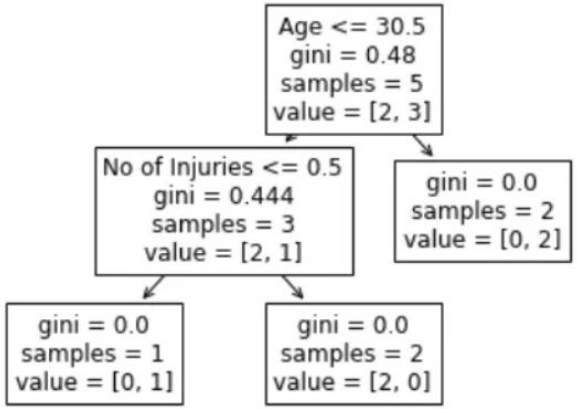
ML Drzewa i Lasy

ccp_alpha



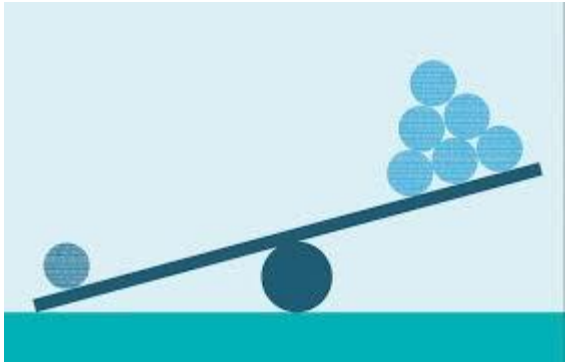
ML Drzewa i Lasy

criterion



ML Drzewa i Lasy

class_weight



infoShareAcademy.com

ML Drzewa i Lasy

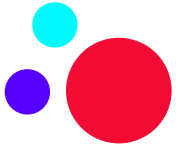
random_state

random_state=42

ML Drzewa i Lasy

Zalety drzew decyzyjnych

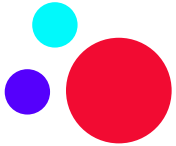
- Łatwa wizualizacja i prosta interpretacja.
- Odpowiedni do problemów klasyfikacji i regresji.
- Niewrażliwość na monotoniczne przekształcenia zmiennych.
- Niewrażliwość na istnienie w algorytmie nieistotnych atrybutów.
- Prosty w obsłudze - można używać cech kategorycznych i liczbowych (uwaga! Sklearn nie obsługuje kategorycznych).



ML Drzewa i Lasy

Wady drzew decyzyjnych

- Niestabilność algorytmu.
- Podatność na overfitting.
- Regresja nie przewiduje danych spoza zakresów, które widziała.
- Podziały ortogonalne (prostopadłe do osi).

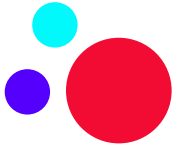


ML Drzewa i Lasy

Implementacja (sklearn)

Drzewa decyzyjne import:

```
from sklearn.datasets import load_iris

from sklearn.tree import (
    DecisionTreeClassifier,
    plot_tree
)

import matplotlib.pyplot as plt

from mlxtend import plotting
```

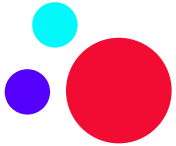


ML Drzewa i Lasy

Implementacja (sklearn)

Problem klasyfikacji:

```
iris = load_iris()

X = iris.data[:, 2:]
y = iris.target
```



ML Drzewa i Lasy

Implementacja (sklearn)

Decision Tree Classifier:

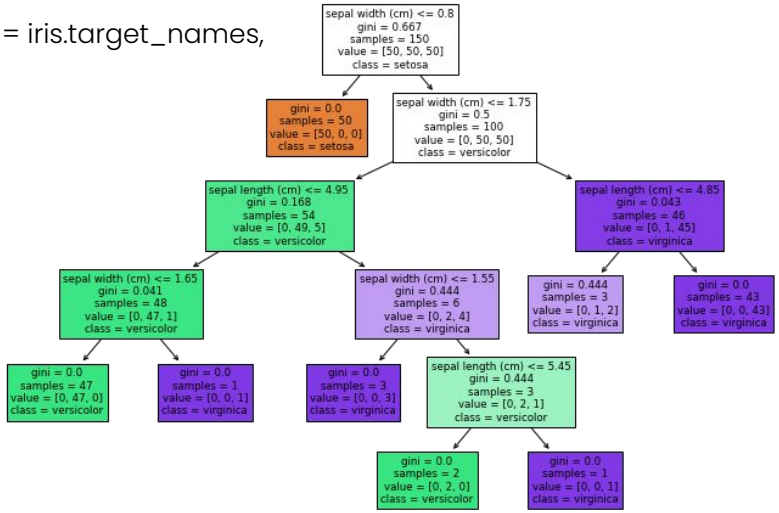
```
tree_clf = DecisionTreeClassifier(random_state=0) #z domyślnymi parametrami

tree_clf.fit(X, y)
```

ML Drzewa i Lasy

Implementacja (sklearn)

```
plt.figure(figsize = (12, 8))

plot_tree(tree_clf,
        feature_names = iris.feature_names,
        class_names = iris.target_names,
        filled=True);
```

Wizualizacja:

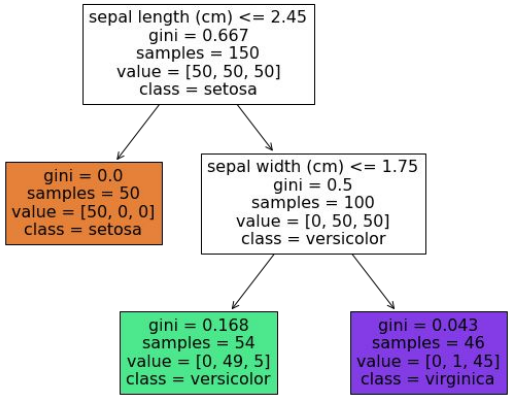


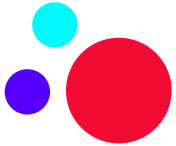
ML Drzewa i Lasy

Implementacja (sklearn)

Przytnijmy trochę drzewo, żeby łatwiej interpretować wyniki.

```
tree_clf = DecisionTreeClassifier(max_depth=2, random_state=99)
tree_clf.fit(X, y)
```





ML Drzewa i Lasy

Implementacja (sklearn)

Istotność zmiennych:

- Ocenia, jak ważna jest każda zmienna dla decyzji podejmowanej przez drzewo.
- Jest to liczba z przedziału od 0 do 1 dla każdej cechy, gdzie 0 oznacza "w ogóle nie używana", a 1 oznacza "doskonale przewiduje target".
- Istotności cech zawsze sumują się do 1.

ML Drzewa i Lasy

Implementacja (sklearn)

Istotność zmiennych:

tree_clf.feature_importances_

array([0.56199095, 0.43800905])

```
for feature, importance in zip(iris.feature_names, importances):
    print(f"{feature}: {importance}")
```

sepal length (cm): 0.5619909502262443

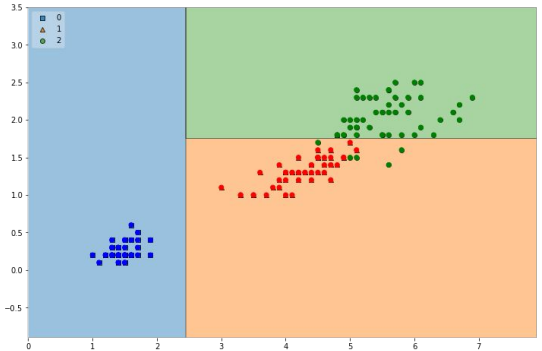
sepal width (cm): 0.4380090497737556

ML Drzewa i Lasy

Implementacja (sklearn)

Granice decyzyjne:

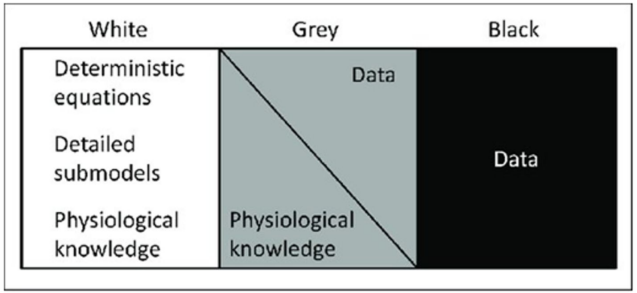
```
def decision_regions(data, target, classifier, figsize=(12, 8)):
    plt.figure(figsize=figsize)
    plotting.plot_decision_regions(X=data, y=target, clf=classifier, legend=2)
    plt.scatter(data[:, 0], data[:, 1], c=["brg"[x] for x in target])

    decision_regions(X, y, tree_clf)
```

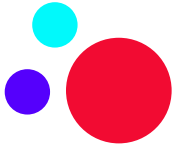


ML Drzewa i Lasy

Implementacja (sklearn)

Interpretacja modelu: White Box Models.





ML Drzewa i Lasy

Implementacja (sklearn)

Interpretacja modelu: White Box Models.

Predykcja: new_obs = [5, 1.5]

```
tree_clf.predict_proba([new_obs])
```

```
array([[0. , 0.90740741, 0.09259259]])
```

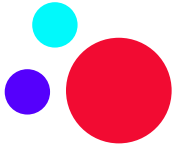
```
tree_clf.predict([[5, 1.5]])
```

```
array([1])
```

```
iris.target_names[1]        'versicolor'
```

Zadanie 13.2 (instrukcja)

1. Dla całości danych Iris (4 featury) zbadaj jak na model wpłynie zmiana kryterium nieczystości z gini na entropy (criterion).
2. Narysuj drzewa.
3. Oblicz samodzielnie wartości gini i entropy w wybranym węźle.
4. Narysuj granice decyzyjne dla drzewa decyzyjnego i regresji logistycznej - w tym celu wybierz podzbiór danych iris iris.data[:,:2].

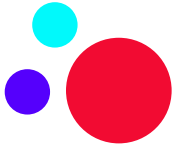


ML Drzewa i Lasy

Implementacja (sklearn)

Problem regresji:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import (
    DecisionTreeRegressor,
    plot_tree
)
from sklearn.linear_model import LinearRegression
```



ML Drzewa i Lasy

Implementacja (sklearn)

Import danych:

```
url =
'https://raw.githubusercontent.com/justmarkham/scikit-learn-v
ideos/master/data/Advertising.csv'
advertising = pd.read_csv(url, index_col=0)

advertising.head()
```

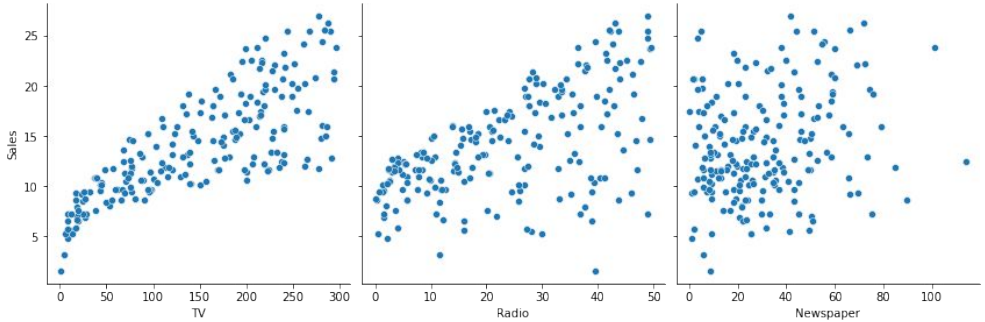
	TV	Radio	Newspaper	Sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9

ML Drzewa i Lasy

Implementacja (sklearn)

Wizualizacja danych:

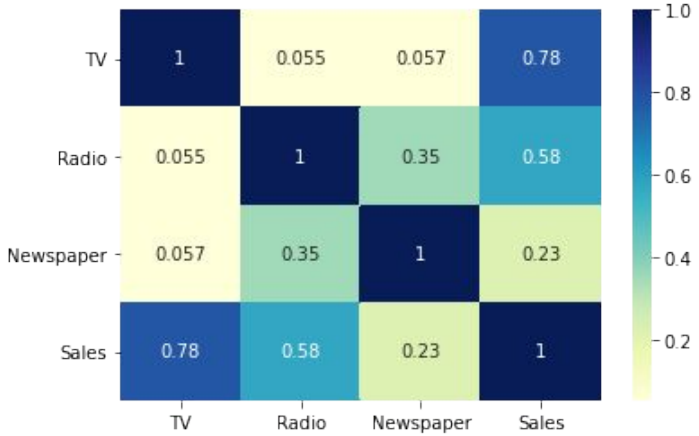
```
sns.pairplot(advertising, x_vars=['TV', 'Radio','Newspaper'],
             y_vars='Sales', height=4, aspect=1, kind='scatter')
plt.show()
```



ML Drzewa i Lasy

Implementacja (sklearn)

```
sns.heatmap(advertising.corr(), cmap="YlGnBu", annot = True)
plt.show()
```



ML Drzewa i Lasy

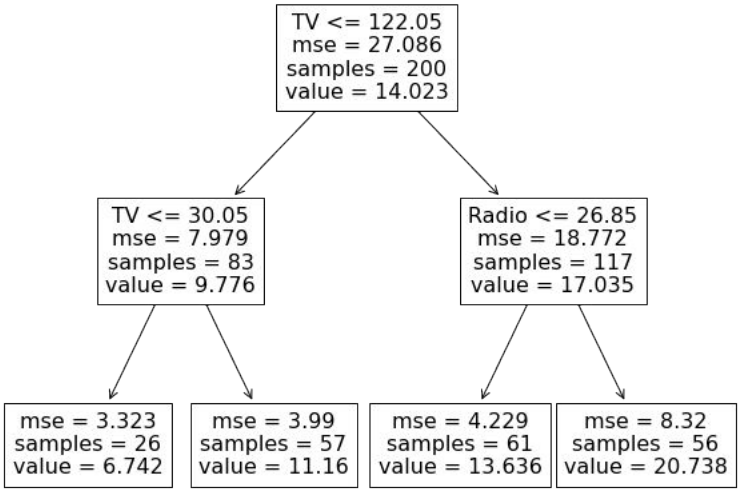
Implementacja (sklearn)

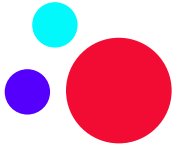
```
feature_names = ['TV','Radio','Newspaper']
model_reg=DecisionTreeRegressor(max_depth=2).fit(advertising[feature_names],
advertising['Sales'])
```

ML Drzewa i Lasy

Implementacja (sklearn)

```
plt.figure(figsize = (10, 8))
plot_tree(model_reg, feature_names = feature_names);
```





ML Drzewa i Lasy

Implementacja (sklearn)

Potencjalny problem:

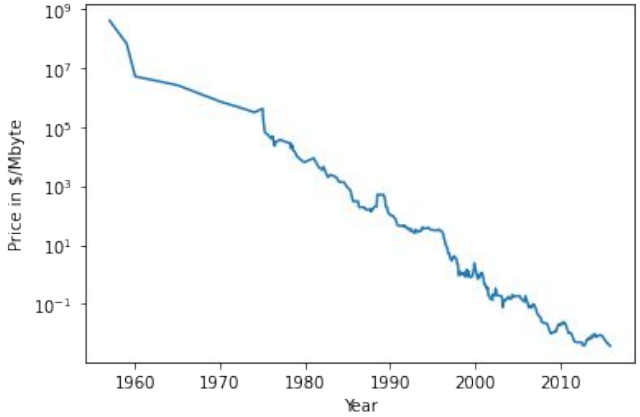
```
url =
https://raw.githubusercontent.com/amueller/introduction_to_ml_with_python/master/data/ram_price.csv'
ram_prices = pd.read_csv(url, index_col=0)
```

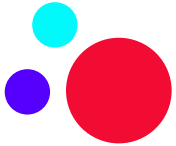
	date	price
0	1957.0	411041792.0
1	1959.0	67947725.0
2	1960.0	5242880.0
3	1965.0	2642412.0
4	1970.0	734003.0

ML Drzewa i Lasy

Implementacja (sklearn)

```
plt.semilogy(ram_prices.date, ram_prices.price)
plt.xlabel("Year")
plt.ylabel("Price in $/Mbyte");
```





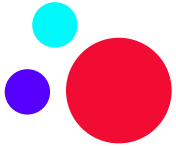
ML Drzewa i Lasy

Implementacja (sklearn)

```
data_train = ram_prices[ram_prices.date < 2000]
data_test = ram_prices[ram_prices.date >= 2000]

X_train = np.array(data_train.date).reshape(-1, 1)
y_train = np.log(data_train.price)
```

dane historyczne

logarytmiczna transformacja



ML Drzewa i Lasy

Implementacja (sklearn)

Porównanie regresji liniowej i drzewa regresyjnego:

```
tree = DecisionTreeRegressor().fit(X_train, y_train)
linear_reg = LinearRegression().fit(X_train, y_train)
```

Predykcja na całym zbiorze:

```
X_all = np.array(ram_prices.date).reshape(-1, 1)
pred_tree = tree.predict(X_all)
pred_lr = linear_reg.predict(X_all)
```

ML Drzewa i Lasy

Implementacja (sklearn)

Porównanie regresji liniowej i drzewa regresyjnego:

Modelowaliśmy transformację logarytmiczną.

Funkcja odwracająca - wykładnicza.

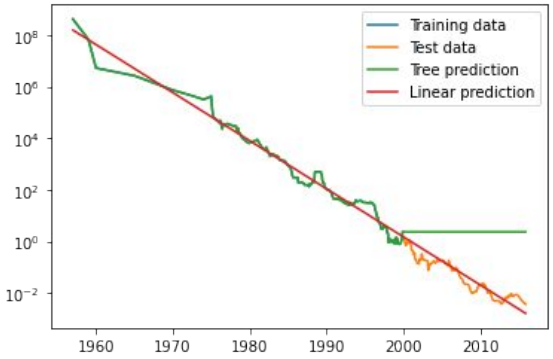
```
price_tree = np.exp(pred_tree)
price_lr = np.exp(pred_lr)
```

ML Drzewa i Lasy

Implementacja (sklearn)

Porównanie regresji liniowej i drzewa regresyjnego:

```
plt.semilogy(data_train.date, data_train.price, label="Training data")
plt.semilogy(data_test.date, data_test.price, label="Test data")
plt.semilogy(ram_prices.date, price_tree, label="Tree prediction")
plt.semilogy(ram_prices.date, price_lr, label="Linear prediction")
plt.legend()
```



ML Drzewa i Lasy

Implementacja (sklearn)

Overfitting:

```
from sklearn.datasets import load_iris

from sklearn.tree import (
    DecisionTreeClassifier,
    DecisionTreeRegressor,
    plot_tree
)

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

from mlxtend import plotting
```

ML Drzewa i Lasy

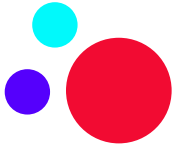
Implementacja (sklearn)

```
def decision_regions(data, target, classifier, figsize=(12, 8)):
    plt.figure(figsize=figsize)
    plotting.plot_decision_regions(X=data, y=target, clf=classifier,
legend=2)
    plt.scatter(data[:, 0], data[:, 1], c=["brg"[x] for x in target])

iris = load_iris()
X = iris.data[:,2:]
y = iris.target
```

ML Drzewa i Lasy

Implementacja (sklearn)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
random_state=42)

tree_clf = DecisionTreeClassifier(random_state=0)
tree_clf.fit(X_train, y_train)

        DecisionTreeClassifier(random_state=0)
```



ML Drzewa i Lasy

Implementacja (sklearn)

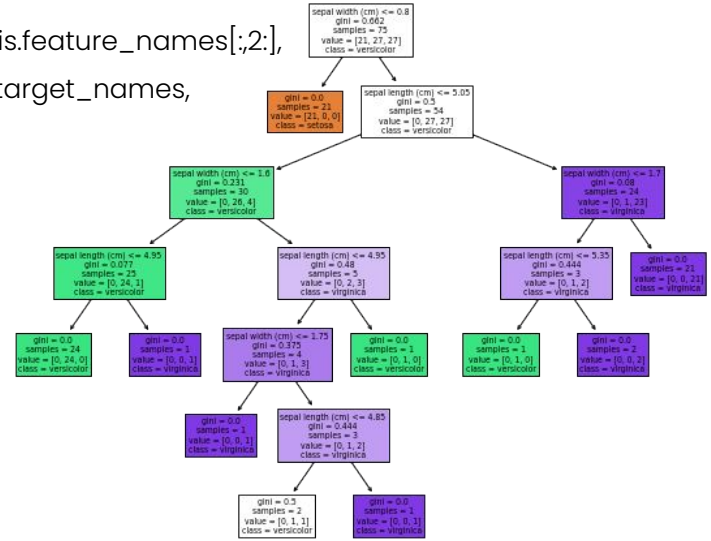

```
print("Accuracy on training set: {:.2f}".format(tree_clf.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(tree_clf.score(X_test, y_test)))
```

```
Accuracy on training set: 0.99
Accuracy on test set: 0.95
```

ML Drzewa i Lasy

Implementacja (sklearn)

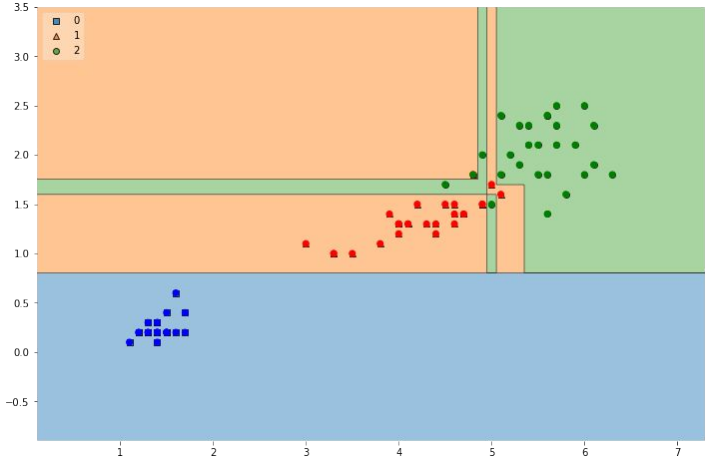
```
plt.figure(figsize = (10, 8))

plot_tree(tree_clf,
        feature_names = iris.feature_names[:,2:],
        class_names = iris.target_names,
        filled=True);
```

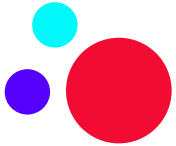


ML Drzewa i Lasy

Implementacja (sklearn)

decision_regions(X_train, y_train, tree_clf)





ML Drzewa i Lasy

Implementacja (sklearn)

Pruning (przycinanie drzewa):

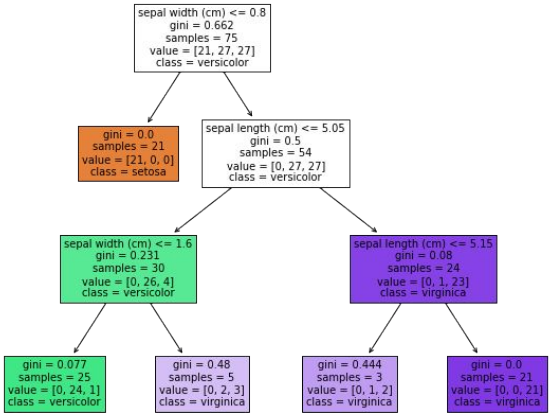
```
tree_clf2 = DecisionTreeClassifier(max_depth=3, random_state=0)
tree_clf2.fit(X_train, y_train)

print("Accuracy on training set: {:.2f}".format(tree_clf2.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(tree_clf2.score(X_test, y_test)))
```

```
Accuracy on training set: 0.95
Accuracy on test set: 1.00
```

ML Drzewa i Lasy

Implementacja (sklearn)

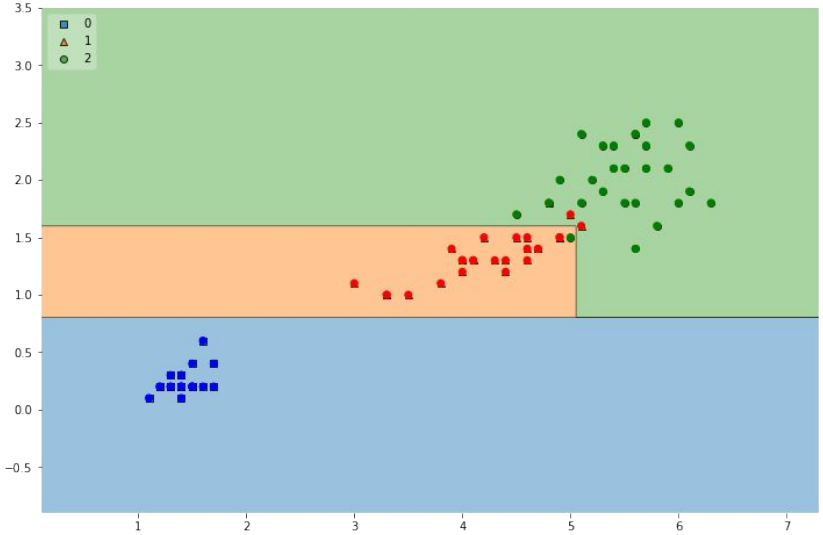
```
plt.figure(figsize = (10, 8))

plot_tree(tree_clf2,
          feature_names = iris.feature_names,
          class_names = iris.target_names,
          filled=True);
```



ML Drzewa i Lasy

Implementacja (sklearn)

decision_regions(X_train, y_train, tree_clf2)



Zadanie 13.3 (instrukcja)

1. Wczytaj zbiór load_breast_cancer dostępny w sklearn.datasets. (UWAGA! w tym przypadku nie da się już zwizualizować za pomocą decision_regions).
2. Przeanalizuj dane.
3. Podziel zbiór na treningowy i testowy w proporcjach 7:3.
4. Wytrenuj model DecisionTreeClassifier bez przycinania.
5. Sprawdź accuracy na zbiorze treningowym i testowym.
6. Narysuj drzewo.
7. Wytrenować 1 model. Sprawdź istotność zmiennych dla najlepszego modelu.

ML Drzewa i Lasy

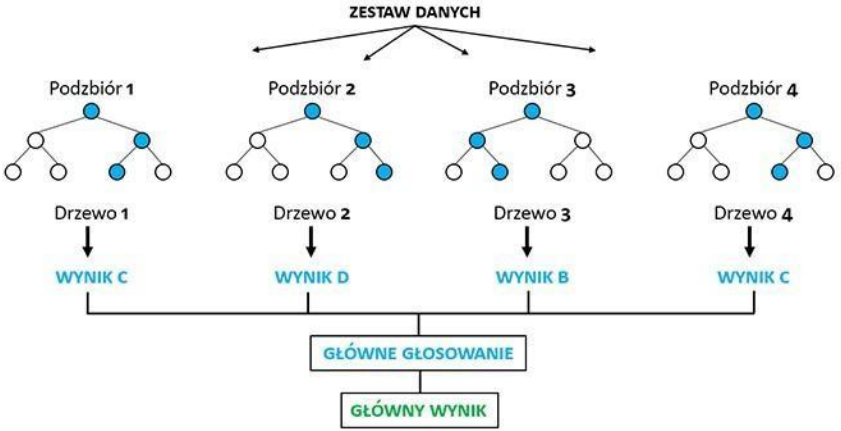
Lasy losowe

- Random Forest to model typu ensemble (komitet klasyfikatorów) czyli model składający się ze zbioru słabszych modeli, których wyniki są następnie przetwarzane (uśredniane lub przeliczane) w celu stworzenia modelu silnego.

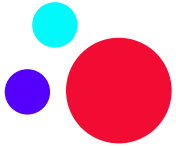
- W przypadku Random Forest podstawowym modelem jest drzewo decyzyjne.

ML Drzewa i Lasy

Lasy losowe



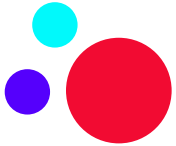
źródło: https://predictivesolutions.pl/jak-udoskonalic-algorytm-drzew-decyzyjnych



ML Drzewa i Lasy

bagging

Bagging to sposób na zmniejszenie variance error.
Zamiast uczyć jedno skomplikowane drzewo uczymy ich wiele wykorzystując technikę **bootstrap**.



ML Drzewa i Lasy

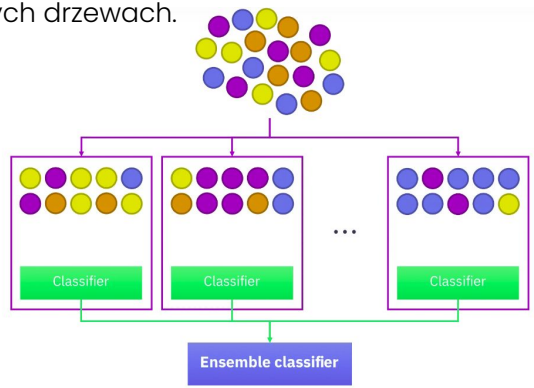
bootstrap

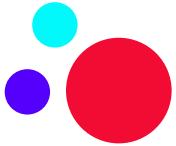
Bootstrap polega na tym, że zamiast uczyć drzewo na danych treningowych uczymy je na tzw. **bootstrap sample**, czyli zestawie danych stworzonych przez losowanie ze zwracaniem z danych treningowych.

ML Drzewa i Lasy

Dlaczego bootstrapujemy?

- Ze względu na różne powtórzenia próbek w zbiorze treningowym oraz pominięcie innych próbek w każdej paczce danych powstałe modele będą skupiały się na różnych aspektach.
- Wartości miar nieczystości zbioru danych będą inne.
- Korzenie drzewa będą dzielić dataset od innych zmiennych.
- Niektóre problematyczne gałęzie będą nieobecne w niektórych drzewach.





ML Drzewa i Lasy

Parametry lasów losowych

n_estimators

max_depth

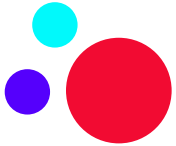
min_samples_leaf

max_features

min_samples_split

criterion

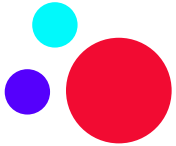
bootstrap



ML Drzewa i Lasy

Zalety lasów losowych

- Efektywna metoda.
- Odpowiednia dla dużych zbiorów.
- Daje oszacowanie, które zmienne są ważne.
- Odpowiednia dla problemów klasyfikacji i regresji.



ML Drzewa i Lasy

Wady lasów losowych

- Potrzebuje większych zasobów.
- Proces decyzyjny bardziej skomplikowany niż w przypadku pojedynczego drzewa - trudniejsze do wytłumaczenia.
- Trudniejsze do wizualizacji od pojedynczego drzewa.

ML Drzewa i Lasy

Implementacja (sklearn)

Problem klasyfikacji

	sepal length(cm)	sepal width(cm)	petal length(cm)	petal width(cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
iris = load_iris()
```

```
pd.DataFrame(iris['data'], columns=iris.feature_names)
```

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

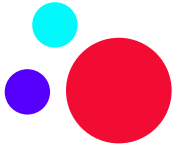
```
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target)
```

ML Drzewa i Lasy

Implementacja (sklearn)

```
rf_classifier = RandomForestClassifier(n_estimators = 5,
criterion = 'gini', max_depth=4, bootstrap=True,
random_state=1)
rf_classifier.fit(X_train, y_train)

rf_classifier.score(X_test, y_test)
```

0.9473684210526315



ML Drzewa i Lasy

Implementacja (sklearn)

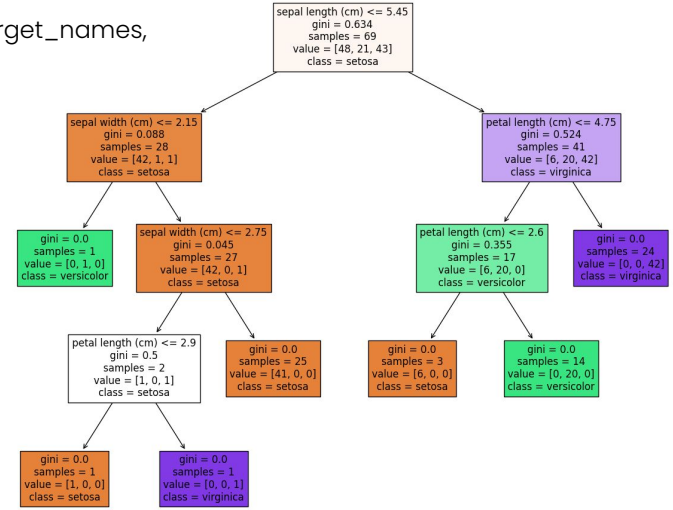
Parametrs estimators_ listuje drzewa decyzyjne, które są wykorzystywane w danym ensemblu. Można je zwizualizować jak każde inne drzewo decyzyjne.

```
rf_classifier.estimators_

[DecisionTreeClassifier(max_depth=4, max_features='sqrt',
                        random_state=1791095845),
 DecisionTreeClassifier(max_depth=4, max_features='sqrt',
                        random_state=2135392491),
 DecisionTreeClassifier(max_depth=4, max_features='sqrt',
random_state=946286476),
 DecisionTreeClassifier(max_depth=4, max_features='sqrt',
                        random_state=1857819720),
 DecisionTreeClassifier(max_depth=4, max_features='sqrt',
random_state=491263)]
```

ML Drzewa i Lasy

Implementacja (sklearn)

```
plt.figure(figsize = (15,12))

plot_tree(rf_classifier.estimators_[0],
          feature_names=iris.feature_names,
          class_names=iris.target_names,
          filled=True);
```



ML Drzewa i Lasy

Implementacja (sklearn)

Lasy losowe pozwalają na uzyskanie oszacowania istotności każdej ze zmiennych.

```
eature_importances = pd.DataFrame(rf_classifier.feature_importances_, index=iris.feature_names,
                                  columns=['importance']).sort_values('importance', ascending=False)
```

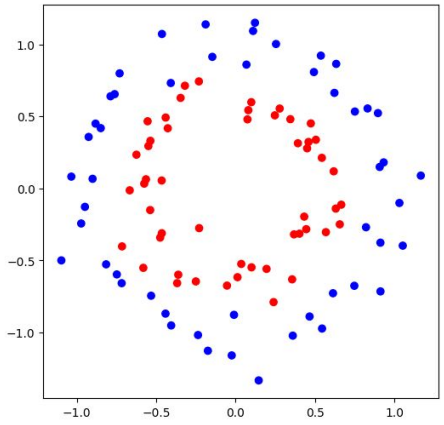
	importance
petal length (cm)	0.365004
petal width (cm)	0.297434
sepal length (cm)	0.239708
sepal width (cm)	0.097855

ML Drzewa i Lasy

Implementacja (sklearn)

Porównanie lasów losowych z drzewami decyzyjnymi:

```
X, y = make_circles(100, noise=0.1, random_state=0, factor=0.6)

plt.figure(figsize=(6, 6))
plt.scatter(X[:, 0], X[:, 1], c=["brg"[x] for x in y])
```



ML Drzewa i Lasy

Implementacja (sklearn)

Porównanie lasów losowych z drzewami decyzyjnymi:

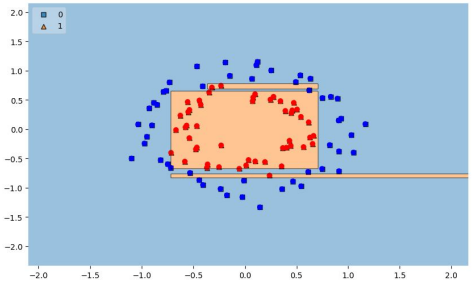
```
def decision_regions(data, target, classifier, figsize=(15, 8)):
    plt.figure(figsize=figsize)
    plotting.plot_decision_regions(X=data, y=target, clf=classifier, legend=2)
    plt.scatter(data[:, 0], data[:, 1], c=["brg"[x] for x in target])

rf_classifier = RandomForestClassifier(n_estimators=10,
criterion='gini', max_depth=7, bootstrap=True,
random_state=1)
tree_classifier = DecisionTreeClassifier(criterion='gini',
max_depth=7)

rf_classifier.fit(X, y)
tree_classifier.fit(X, y)
```

ML Drzewa i Lasy

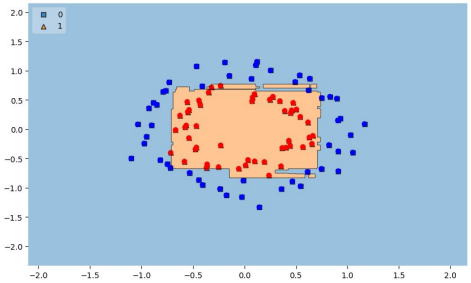
Implementacja (sklearn)

Porównanie lasów losowych z drzewami decyzyjnymi:

Drzewo decyzyjne



Lasy losowe

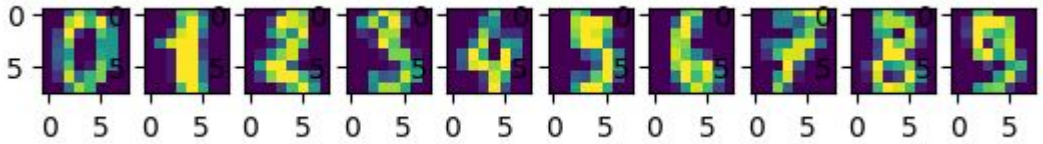


Zarówno lasy jak i drzewa decyzyjne mają podobną, "kanciastą" granicę decyzyjną z pewną ilością wysp, próbującą się dopasować możliwie dobrze do zestawu treningowego.

ML Drzewa i Lasy

Implementacja (sklearn)

```
digits = load_digits()

fig, axes = plt.subplots(1, 10)

for i in range(10):
    axes[i].imshow(digits.data[i].reshape(8, 8))
```

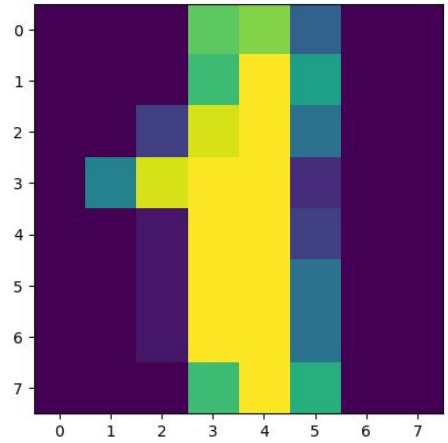


ML Drzewa i Lasy

Implementacja (sklearn)

```
digits.data[1].reshape(8, 8)
```

```
array([[ 0.,  0.,  0., 12., 13.,  5.,  0.,  0.],
       [ 0.,  0.,  0., 11., 16.,  9.,  0.,  0.],
       [ 0.,  0.,  3., 15., 16.,  6.,  0.,  0.],
       [ 0.,  7., 15., 16., 16.,  2.,  0.,  0.],
       [ 0.,  0.,  1., 16., 16.,  3.,  0.,  0.],
       [ 0.,  0.,  1., 16., 16.,  6.,  0.,  0.],
       [ 0.,  0.,  1., 16., 16.,  6.,  0.,  0.],
       [ 0.,  0.,  0., 11., 16., 10.,  0.,  0.]])
```

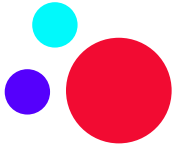


ML Drzewa i Lasy

Implementacja (sklearn)

```
digits.target

    array([0, 1, 2, ..., 8, 9, 8])/
```

```
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, random_state=0)

rf_classifier = RandomForestClassifier(n_estimators=10,
criterion='gini', max_depth=10, bootstrap=True,
random_state=1)
tree_classifier = DecisionTreeClassifier(criterion='gini',
max_depth=10)

rf_classifier.fit(X_train, y_train)
tree_classifier.fit(X_train, y_train);
```



ML Drzewa i Lasy

Implementacja (sklearn)


```
tree_classifier.score(X_train, y_train),
tree_classifier.score(X_test, y_test)
```

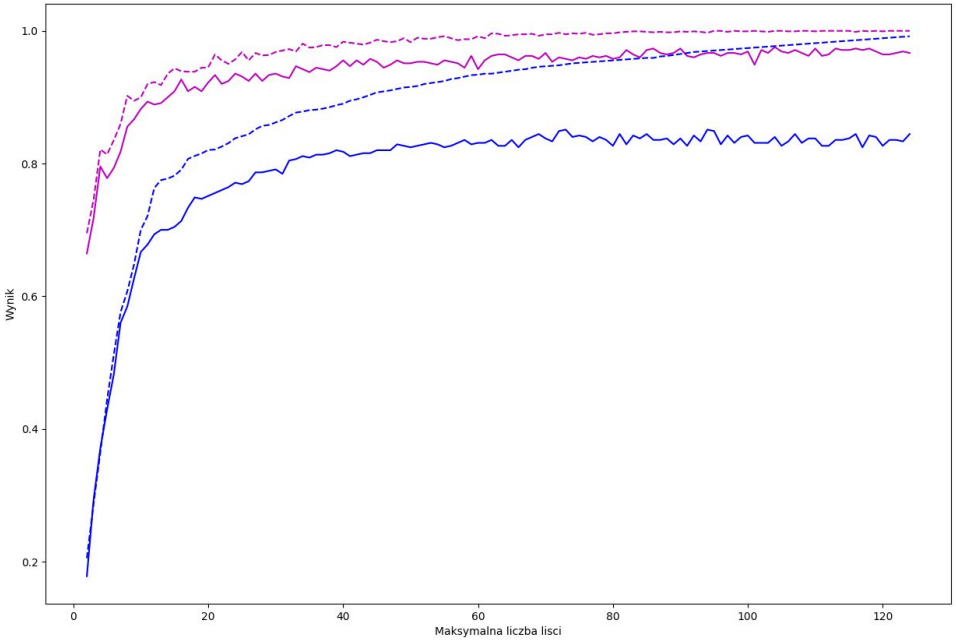
```
(0.9784706755753526, 0.8333333333333334)
```

```
rf_classifier.score(X_train, y_train),
rf_classifier.score(X_test, y_test)
```

```
(0.9985152190051967, 0.9155555555555556)
```

ML Drzewa i Lasy

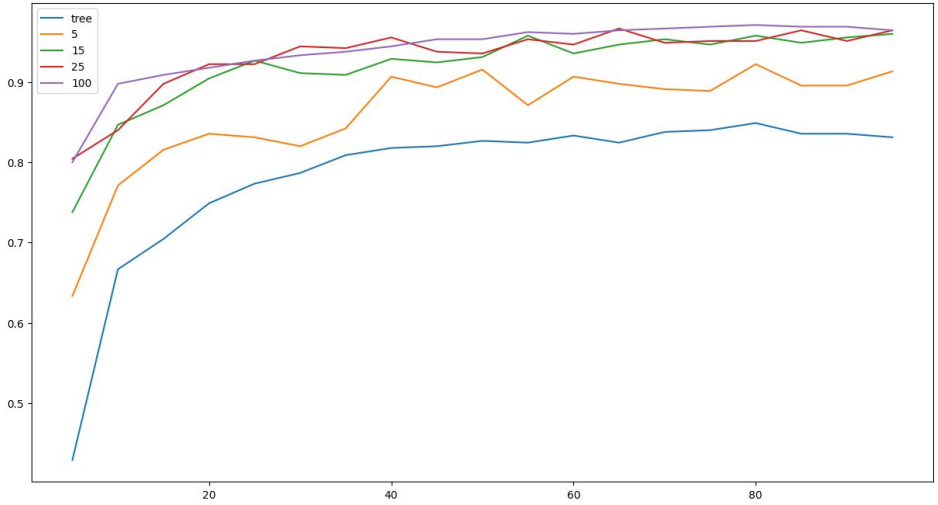
Implementacja (sklearn)



infoShareAcademy.com

ML Drzewa i Lasy

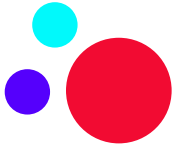
Implementacja (sklearn)



Zadanie 13.4 (instrukcja)

Wykreśl zależność między min_samples_leaf i max_depth a dokładnością na zbiorze testowym.
(Analogiczny wykres jak ten z max_leaf_nodes i n_estimators).
Użyj poniższego zestawu danych generowanego przez funkcję datasets.make_moons.
Do oceny dokładności użyj metody .score().

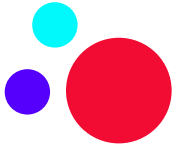
X, y = make_moons(n_samples=500, noise=0.1)



ML Drzewa i Lasy

Problem regresji


```
%matplotlib inline
from sklearn.ensemble import RandomForestRegressor
from sklearn import tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.datasets import load_boston
```

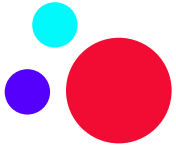


ML Drzewa i Lasy

Implementacja (sklearn)


```
boston = load_boston()
all_x, all_y = boston.data, boston.target

rf_regressor = RandomForestRegressor()
rf_regressor.fit(all_x, all_y)

rf_regressor.score(all_x, all_y)
```

```
0.9836237806157219
```



ML Drzewa i Lasy

Implementacja (sklearn)

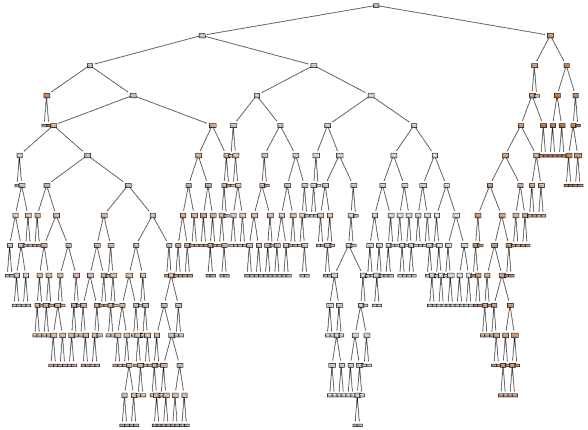

```
train_x, test_x, train_y, test_y = train_test_split(all_x, all_y, test_size=1/3, random_state=42)

rf_regressor = RandomForestRegressor()
rf_regressor.fit(train_x, train_y)

cross_val_score(rf_regressor, train_x, train_y, cv=5).mean()
```

```
0.8132878349182103
```

ML Drzewa i Lasy

Implementacja (sklearn)

```
plt.figure(figsize = (15,12))

plot_tree(
    rf_regressor.estimators_[2],
    filled=True,
        );
```



ML Drzewa i Lasy

Podsumowanie