**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика с системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Технологии машинного обучения»

Отчёт по лабораторной работе №4

Выполнил: Проверил:

студент группы РТ5-61Б преподаватель каф. ИУ5

Кузнецов А.В. Гапанюк Ю.Е.

Подпись и дата: Подпись и дата:

2023 г.

# Линейные модели, SVM и деревья решений.

### Описание датасета

Датасет качества вина. Выбран, потому что решает задачи классификации.

import numpy as np  
import pandas as pd  
from typing import Dict, Tuple  
import matplotlib.pyplot as plt  
from scipy import stats  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier  
from sklearn.metrics import accuracy\_score, balanced\_accuracy\_score  
from sklearn.metrics import precision\_score, recall\_score, f1\_score, classification\_report  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, mean\_squared\_log\_error, median\_absolute\_error, r2\_score   
from sklearn.metrics import roc\_curve, roc\_auc\_score  
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export\_graphviz  
import seaborn as sns  
from sklearn import preprocessing  
from sklearn import datasets  
from sklearn import utils  
  
ds = datasets.load\_iris()  
ds.data[:5]  
np.unique(ds.target)  
iris\_df = pd.DataFrame(data= np.c\_[ds['data'], ds['target']],  
 columns= ds['feature\_names'] + ['target'])  
iris\_df.describe()

sepal length (cm) sepal width (cm) petal length (cm) \  
count 150.000000 150.000000 150.000000   
mean 5.843333 3.057333 3.758000   
std 0.828066 0.435866 1.765298   
min 4.300000 2.000000 1.000000   
25% 5.100000 2.800000 1.600000   
50% 5.800000 3.000000 4.350000   
75% 6.400000 3.300000 5.100000   
max 7.900000 4.400000 6.900000   
  
 petal width (cm) target   
count 150.000000 150.000000   
mean 1.199333 1.000000   
std 0.762238 0.819232   
min 0.100000 0.000000   
25% 0.300000 0.000000   
50% 1.300000 1.000000   
75% 1.800000 2.000000   
max 2.500000 2.000000

В датасете нет пустых элементов

### С использованием метода train\_test\_split разделите выборку на обучающую и тестовую.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 ds.data, ds.target, test\_size=0.5, random\_state=1)

Размер обучающей модели

X\_train.shape, y\_train.shape

((75, 4), (75,))

Размер тестовой выборки

X\_test.shape, y\_test.shape

((75, 4), (75,))

Функция train\_test\_split разделила исходную выборку таким образом, чтобы в обучающей и тестовой частях сохранились все классы.

np.unique(y\_train)  
np.unique(y\_test)

array([0, 1, 2])

def class\_proportions(array: np.ndarray) -> Dict[int, Tuple[int, float]]:  
 labels, counts = np.unique(array, return\_counts=True)  
 counts\_perc = counts/array.size  
 res = dict()  
 for label, count2 in zip(labels, zip(counts, counts\_perc)):  
 res[label] = count2  
 return res  
  
def print\_class\_proportions(array: np.ndarray):  
 proportions = class\_proportions(array)  
 if len(proportions)>0:  
 print('Метка \t Количество \t Процент встречаемости')  
 for i in proportions:  
 val, val\_perc = proportions[i]  
 val\_perc\_100 = round(val\_perc \* 100, 2)  
 print('{} \t {} \t \t {}%'.format(i, val, val\_perc\_100))

В исходной выборке нет явного дисбаланса классов для целевого признака

print\_class\_proportions(ds.target)

Метка Количество Процент встречаемости  
0 50 33.33%  
1 50 33.33%  
2 50 33.33%

Функция train\_test\_split разделила исходную выборку таким образом, чтобы в обучающей и тестовой частях сохранились пропорции классов.

print\_class\_proportions(y\_train)

Метка Количество Процент встречаемости  
0 26 34.67%  
1 26 34.67%  
2 23 30.67%

Для обучающей выборки

print\_class\_proportions(y\_test)

Метка Количество Процент встречаемости  
0 24 32.0%  
1 24 32.0%  
2 27 36.0%

Для тестовой выборки

lab = preprocessing.LabelEncoder()  
cl1\_1 = KNeighborsClassifier(n\_neighbors=2)  
y\_transformed = lab.fit\_transform(y\_train)  
cl1\_1.fit(X\_train, y\_transformed)  
target1\_1 = cl1\_1.predict(X\_test)  
len(target1\_1), target1\_1

(75,  
 array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,  
 1, 0, 2, 1, 0, 0, 1, 1, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 1, 2,  
 1, 2, 0, 0, 0, 1, 0, 0, 2, 2, 2, 2, 1, 1, 2, 1, 0, 2, 1, 0, 0, 2,  
 0, 1, 2, 1, 1, 2, 1, 0, 1]))

2 ближайших соседа

lab = preprocessing.LabelEncoder()  
cl1\_1 = KNeighborsClassifier(n\_neighbors=10)  
y\_transformed = lab.fit\_transform(y\_train)  
cl1\_1.fit(X\_train, y\_transformed)  
target1\_2 = cl1\_1.predict(X\_test)  
len(target1\_2), target1\_2

(75,  
 array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,  
 2, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 1, 2,  
 1, 2, 0, 0, 0, 1, 0, 0, 2, 2, 2, 2, 2, 1, 2, 1, 0, 2, 2, 0, 0, 2,  
 0, 2, 2, 1, 1, 2, 2, 0, 1]))

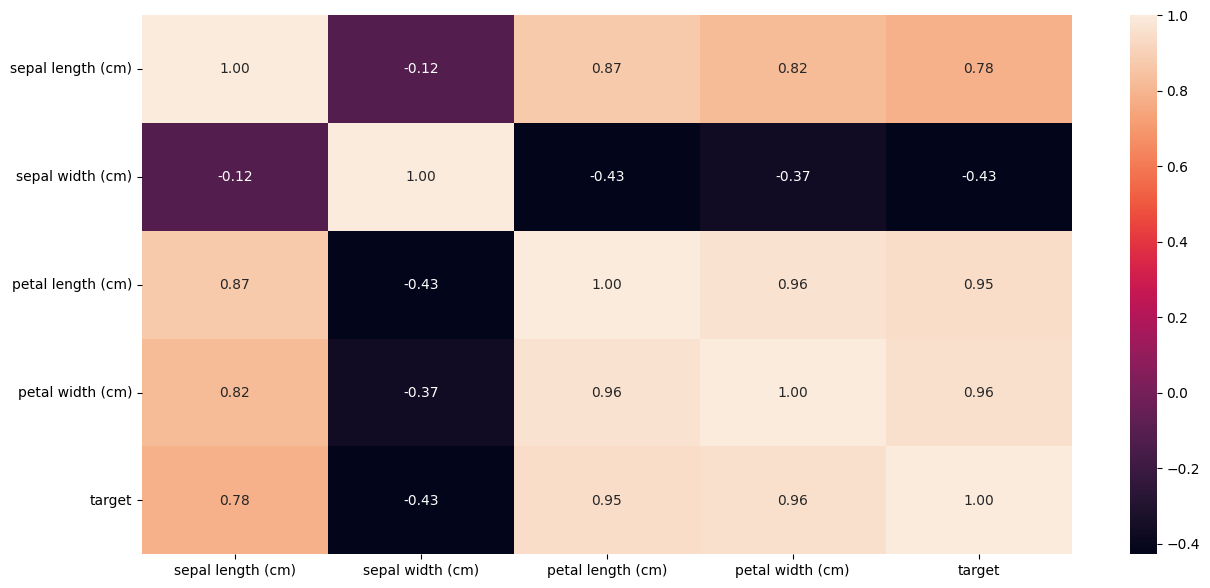
10 ближайших соседей

## Обучить одну из линейных моделей (линейную или полиномиальную регрессию при решении задачи регрессии, логистическую регрессию при решении задачи классификации);

Построим корреляционную матрицу

fig, ax = plt.subplots(figsize=(15,7))  
sns.heatmap(iris\_df.corr(method='pearson'), ax=ax, annot=True, fmt='.2f')

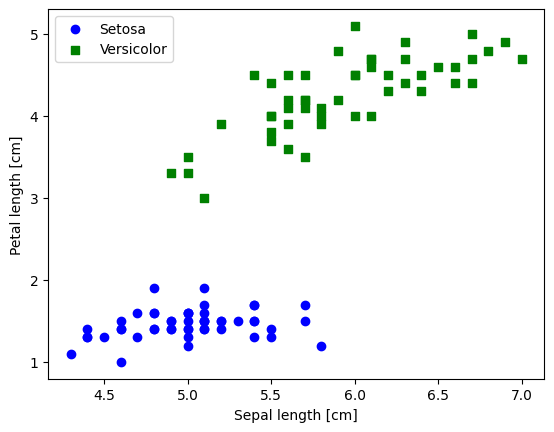
<AxesSubplot:>



y = iris\_df.iloc[0:100, 4].values  
y = np.where(y == 'Iris-setosa', 0, 1)  
   
  
X = iris\_df.iloc[0:100, [0, 2]].values  
  
plt.scatter(X[:50, 0], X[:50, 1],  
 color='blue', marker='o', label='Setosa')  
plt.scatter(X[50:100, 0], X[50:100, 1],  
 color='green', marker='s', label='Versicolor')  
   
plt.xlabel('Sepal length [cm]')  
plt.ylabel('Petal length [cm]')  
plt.legend(loc='upper left')

/var/folders/k2/sflz85vs63j4gxgb780kr\_hr0000gn/T/ipykernel\_29438/220640260.py:2: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison  
 y = np.where(y == 'Iris-setosa', 0, 1)

<matplotlib.legend.Legend at 0x1302d5f10>



## Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

### Accuracy

Метрика вычисляет процент (долю в диапазоне от 0 до 1) правильно определенных классов.

Эту метрику обычно переводят как "точность", но перевод не является удачным, потому что совпадает с переводом для другой метрики - "precision".

Чтобы не сталкиваться с неточностями перевода, названия метрик можно не переводить.

# 2 ближайших соседа  
accuracy\_score(y\_test, target1\_1)

0.92

# 10 ближайших соседей  
accuracy\_score(y\_test, target1\_2)

0.9733333333333334

Точность в случае 10 ближайших соседей составляет 97%, а точность в случае 2 ближайших соседей составляет 92%.

Метрика "Accuracy" показывает точность по всем классам, но точность может быть различной для различных классов.

Это очень серьезная проблема, которая часто возникает на несбалансированных выборках.

def accuracy\_score\_for\_classes(  
 y\_true: np.ndarray,   
 y\_pred: np.ndarray) -> Dict[int, float]:  
 d = {'t': y\_true, 'p': y\_pred}  
 df = pd.DataFrame(data=d)  
 # Метки классов  
 classes = np.unique(y\_true)  
 # Результирующий словарь  
 res = dict()  
 # Перебор меток классов  
 for c in classes:  
 # отфильтруем данные, которые соответствуют   
 # текущей метке класса в истинных значениях  
 temp\_data\_flt = df[df['t']==c]  
 # расчет accuracy для заданной метки класса  
 temp\_acc = accuracy\_score(  
 temp\_data\_flt['t'].values,   
 temp\_data\_flt['p'].values)  
 # сохранение результата в словарь  
 res[c] = temp\_acc  
 return res  
  
def print\_accuracy\_score\_for\_classes(  
 y\_true: np.ndarray,   
 y\_pred: np.ndarray):  
 accs = accuracy\_score\_for\_classes(y\_true, y\_pred)  
 if len(accs)>0:  
 print('Метка \t Accuracy')  
 for i in accs:  
 print('{} \t {}'.format(i, accs[i]))

# 2 ближайших соседа  
print\_accuracy\_score\_for\_classes(y\_test, target1\_1)

Метка Accuracy  
0 1.0  
1 1.0  
2 0.7777777777777778

# 10 ближайших соседей  
print\_accuracy\_score\_for\_classes(y\_test, target1\_2)

Метка Accuracy  
0 1.0  
1 0.9583333333333334  
2 0.9629629629629629

# Конвертация целевого признака в бинарный  
def convert\_target\_to\_binary(array:np.ndarray, target:int) -> np.ndarray:  
 # Если целевой признак совпадает с указанным, то 1 иначе 0  
 res = [1 if x==target else 0 for x in array]  
 return res

bin\_iris\_y\_train = convert\_target\_to\_binary(y\_train, 2)  
list(zip(y\_train, bin\_iris\_y\_train))[:10]

[(1, 0),  
 (2, 1),  
 (1, 0),  
 (2, 1),  
 (1, 0),  
 (0, 0),  
 (0, 0),  
 (0, 0),  
 (2, 1),  
 (0, 0)]

bin\_iris\_y\_test = convert\_target\_to\_binary(y\_test, 2)  
list(zip(y\_train, bin\_iris\_y\_test))[:10]

[(1, 0),  
 (2, 0),  
 (1, 0),  
 (2, 0),  
 (1, 1),  
 (0, 0),  
 (0, 1),  
 (0, 0),  
 (2, 0),  
 (0, 1)]

bin\_target1\_1 = convert\_target\_to\_binary(target1\_1, 2)  
bin\_target1\_2 = convert\_target\_to\_binary(target1\_2, 2)

balanced\_accuracy\_score(bin\_iris\_y\_test, bin\_target1\_1)

0.8888888888888888

balanced\_accuracy\_score(bin\_iris\_y\_test, bin\_target1\_2)

0.9710648148148148

Метрика Accuracy интуитивно понятна и часто используется на практике. Но если количество классов относительно невелико, то лучше всего вычислять Accuracy отдельно для каждого класса.

### Precision, recall и F-мера

# По умолчанию метрики считаются для 1 класса бинарной классификации  
# Для 2 ближайших соседей  
precision\_score(bin\_iris\_y\_test, bin\_target1\_1), recall\_score(bin\_iris\_y\_test, bin\_target1\_1)

(1.0, 0.7777777777777778)

# Для 10 ближайших соседей  
precision\_score(bin\_iris\_y\_test, bin\_target1\_2), recall\_score(bin\_iris\_y\_test, bin\_target1\_2)

(0.9629629629629629, 0.9629629629629629)

Параметр average (уровень детализации метрик) описан в документации к функциям. Рассмотрим данный параметр на примере

# Параметры TP, TN, FP, FN считаются как сумма по всем классам  
precision\_score(y\_test, target1\_1, average='micro')

0.92

# Параметры TP, TN, FP, FN считаются отдельно для каждого класса   
# и берется среднее значение, дисбаланс классов не учитывается.  
precision\_score(y\_test, target1\_1, average='macro')

0.9333333333333332

# Параметры TP, TN, FP, FN считаются отдельно для каждого класса   
# и берется средневзвешенное значение, дисбаланс классов учитывается   
# в виде веса классов (вес - количество истинных значений каждого класса).  
precision\_score(y\_test, target1\_1, average='weighted')

0.936

f1\_score(bin\_iris\_y\_test, bin\_target1\_2)

0.9629629629629629

f1\_score(y\_test, target1\_1, average='micro')

0.92

f1\_score(y\_test, target1\_1, average='macro')

0.9212962962962963

f1\_score(y\_test, target1\_1, average='weighted')

0.9194444444444446

Функция classification\_report позволяет выводить значения точности, полноты и F-меры для всех классов выборки.

classification\_report(y\_test, target1\_1,   
 target\_names=ds.target\_names, output\_dict=True)

{'setosa': {'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 24},  
 'versicolor': {'precision': 0.8,  
 'recall': 1.0,  
 'f1-score': 0.888888888888889,  
 'support': 24},  
 'virginica': {'precision': 1.0,  
 'recall': 0.7777777777777778,  
 'f1-score': 0.8750000000000001,  
 'support': 27},  
 'accuracy': 0.92,  
 'macro avg': {'precision': 0.9333333333333332,  
 'recall': 0.9259259259259259,  
 'f1-score': 0.9212962962962963,  
 'support': 75},  
 'weighted avg': {'precision': 0.936,  
 'recall': 0.92,  
 'f1-score': 0.9194444444444446,  
 'support': 75}}

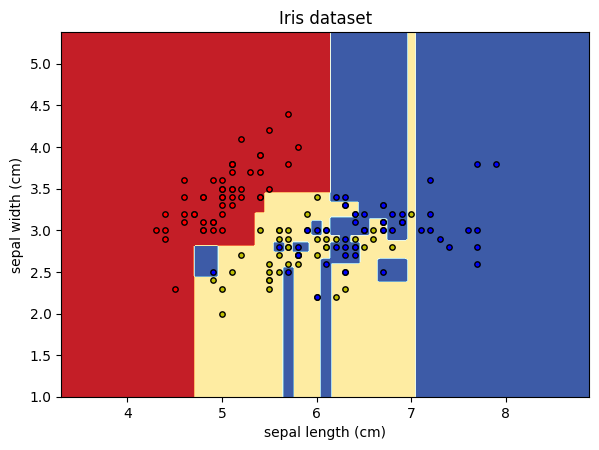
## Постройте график, показывающий важность признаков в дереве решений.

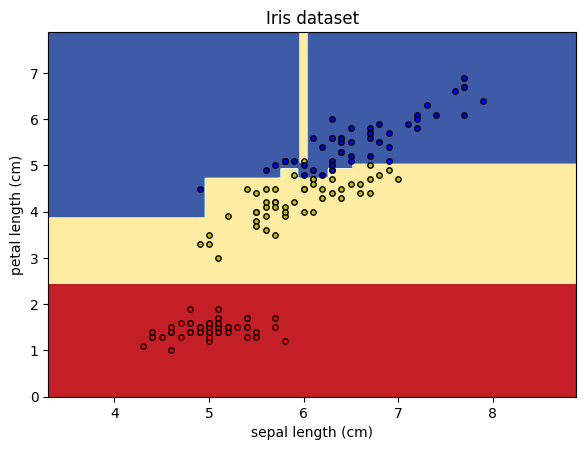
С помощью свойства feature\_importances\_ можно получить значение важности признаков.

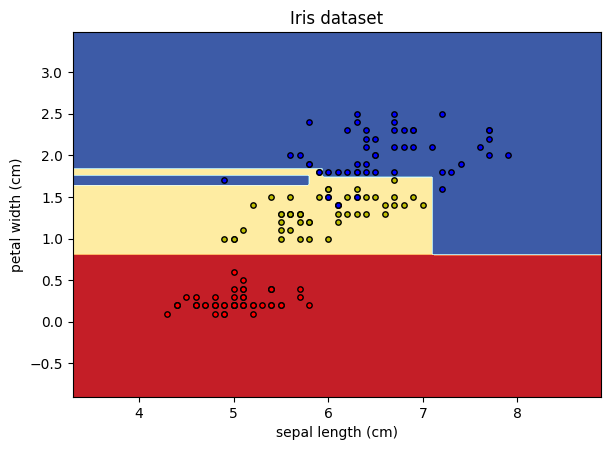
Вычисление важности признаков основано на том, какое количество раз признак встречается в условиях дерева. Чем чаще встречается признак, тем более он важен.

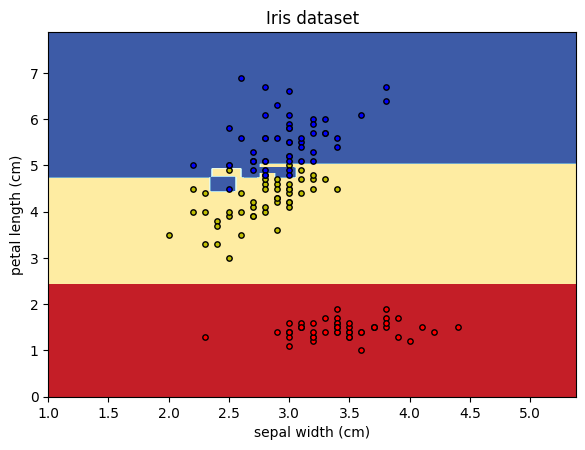
def plot\_tree\_classification(title\_param, ds):  
 """  
 Построение деревьев и вывод графиков для заданного датасета  
 """  
   
 n\_classes = len(np.unique(ds.target))  
 plot\_colors = "ryb"  
 plot\_step = 0.02  
   
 for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],  
 [1, 2], [1, 3], [2, 3]]):  
 # We only take the two corresponding features  
 X = ds.data[:, pair]  
 y = ds.target  
  
 # Train  
 clf = DecisionTreeClassifier(random\_state=1).fit(X, y)  
  
 plt.title(title\_param)  
  
 x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  
 y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  
 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, plot\_step),  
 np.arange(y\_min, y\_max, plot\_step))  
 plt.tight\_layout(h\_pad=0.5, w\_pad=0.5, pad=2.5)  
  
 Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])  
 Z = Z.reshape(xx.shape)  
 cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)  
  
 plt.xlabel(ds.feature\_names[pair[0]])  
 plt.ylabel(ds.feature\_names[pair[1]])  
  
 # Plot the training points  
 for i, color in zip(range(n\_classes), plot\_colors):  
 idx = np.where(y == i)  
 plt.scatter(X[idx, 0], X[idx, 1], c=color, label=ds.target\_names[i],  
 cmap=plt.cm.RdYlBu, edgecolor='black', s=15)  
  
 plt.show()

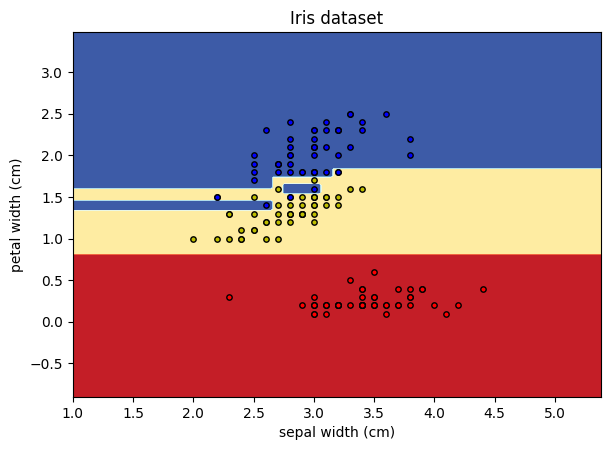
plot\_tree\_classification('Iris dataset', ds)

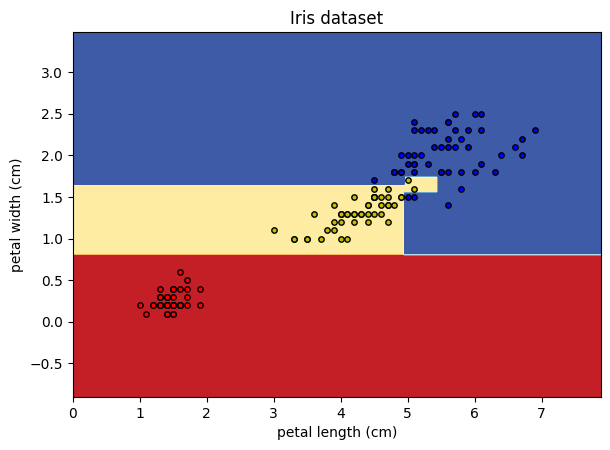












iris\_tree\_cl = DecisionTreeClassifier(random\_state=1)  
iris\_tree\_cl.fit(iris\_df, ds.target)  
iris\_tree\_cl

DecisionTreeClassifier(random\_state=1)

list(zip(iris\_df.columns.values, iris\_tree\_cl.feature\_importances\_))

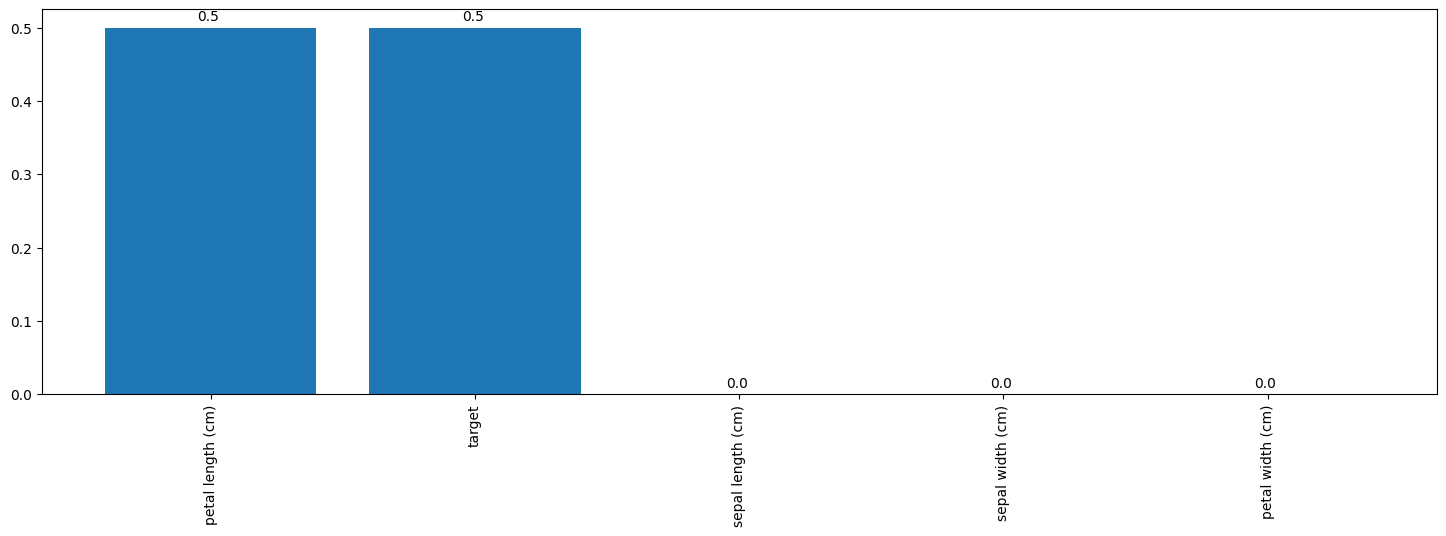
[('sepal length (cm)', 0.0),  
 ('sepal width (cm)', 0.0),  
 ('petal length (cm)', 0.5000000000000001),  
 ('petal width (cm)', 0.0),  
 ('target', 0.49999999999999994)]

# Важность признаков в сумме дает единицу  
sum(iris\_tree\_cl.feature\_importances\_)

1.0

from operator import itemgetter  
  
def draw\_feature\_importances(tree\_model, X\_dataset, figsize=(18,5)):  
 """  
 Вывод важности признаков в виде графика  
 """  
 # Сортировка значений важности признаков по убыванию  
 list\_to\_sort = list(zip(X\_dataset.columns.values, tree\_model.feature\_importances\_))  
 sorted\_list = sorted(list\_to\_sort, key=itemgetter(1), reverse = True)  
 # Названия признаков  
 labels = [x for x,\_ in sorted\_list]  
 # Важности признаков  
 data = [x for \_,x in sorted\_list]  
 # Вывод графика  
 fig, ax = plt.subplots(figsize=figsize)  
 ind = np.arange(len(labels))  
 plt.bar(ind, data)  
 plt.xticks(ind, labels, rotation='vertical')  
 # Вывод значений  
 for a,b in zip(ind, data):  
 plt.text(a-0.05, b+0.01, str(round(b,3)))  
 plt.show()  
 return labels, data

iris\_tree\_cl\_fl, iris\_tree\_cl\_fd = draw\_feature\_importances(iris\_tree\_cl, iris\_df)



iris\_tree\_cl\_fl, iris\_tree\_cl\_fd

(['petal length (cm)',  
 'target',  
 'sepal length (cm)',  
 'sepal width (cm)',  
 'petal width (cm)'],  
 [0.5000000000000001, 0.49999999999999994, 0.0, 0.0, 0.0])

iris\_df.head()

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   
  
 target   
0 0.0   
1 0.0   
2 0.0   
3 0.0   
4 0.0

# Пересортируем признаки на основе важности  
iris\_x\_ds\_sorted = iris\_df[iris\_tree\_cl\_fl]  
iris\_x\_ds\_sorted.head()

petal length (cm) target sepal length (cm) sepal width (cm) \  
0 1.4 0.0 5.1 3.5   
1 1.4 0.0 4.9 3.0   
2 1.3 0.0 4.7 3.2   
3 1.5 0.0 4.6 3.1   
4 1.4 0.0 5.0 3.6   
  
 petal width (cm)   
0 0.2   
1 0.2   
2 0.2   
3 0.2   
4 0.2

# Разделим выборку на обучающую и тестовую  
iris\_X\_train, iris\_X\_test, iris\_y\_train, iris\_y\_test = train\_test\_split(  
 iris\_x\_ds\_sorted, ds.target, test\_size=0.5, random\_state=1)  
iris\_X\_train.shape, iris\_X\_test.shape

((75, 5), (75, 5))

# Обучим дерево и предскажем результаты на всех признаках   
iris\_tree\_cl\_feat\_1 = DecisionTreeClassifier(random\_state=1).fit(iris\_X\_train, iris\_y\_train)  
iris\_y\_test\_predict = iris\_tree\_cl\_feat\_1.predict(iris\_X\_test)  
iris\_y\_test\_predict.shape

(75,)

# Проверим точность по классам  
print\_accuracy\_score\_for\_classes(iris\_y\_test, iris\_y\_test\_predict)

Метка Accuracy  
0 1.0  
1 1.0  
2 1.0

# Обучим дерево и предскажем результаты на единственном самом важном признаке   
iris\_tree\_cl\_feat\_2 = DecisionTreeClassifier(random\_state=1).fit(iris\_X\_train[[iris\_tree\_cl\_fl[0]]], iris\_y\_train)  
iris\_y\_test\_predict\_2 = iris\_tree\_cl\_feat\_2.predict(iris\_X\_test[[iris\_tree\_cl\_fl[0]]])  
iris\_y\_test\_predict\_2.shape

(75,)

# Проверим точность по классам  
print\_accuracy\_score\_for\_classes(iris\_y\_test, iris\_y\_test\_predict\_2)

Метка Accuracy  
0 1.0  
1 1.0  
2 0.7407407407407407

## Визуализируйте дерево решений или выведите правила дерева решений в текстовом виде.

from IPython.core.display import HTML  
from sklearn.tree import export\_text  
tree\_rules = export\_text(iris\_tree\_cl, feature\_names=list(iris\_df.columns))  
HTML('<pre>' + tree\_rules + '</pre>')

<IPython.core.display.HTML object>

iris\_tree\_cl = DecisionTreeClassifier(random\_state=1)  
iris\_tree\_cl.fit(iris\_df, ds.target)  
dot\_data = export\_graphviz(iris\_tree\_cl, out\_file=None,   
 feature\_names=ds.feature\_names,   
 class\_names=ds.target\_names,   
 filled=True, rounded=True, special\_characters=True)   
graph = graphviz.Source(dot\_data)   
graph

---------------------------------------------------------------------------  
ValueError Traceback (most recent call last)  
Cell In[51], line 3  
 1 iris\_tree\_cl = DecisionTreeClassifier(random\_state=1)  
 2 iris\_tree\_cl.fit(iris\_df, ds.target)  
----> 3 dot\_data = export\_graphviz(iris\_tree\_cl, out\_file=None,   
 4 feature\_names=ds.feature\_names,   
 5 class\_names=ds.target\_names,   
 6 filled=True, rounded=True, special\_characters=True)   
 7 graph = graphviz.Source(dot\_data)   
 8 graph  
  
File /usr/local/lib/python3.9/site-packages/sklearn/tree/\_export.py:888, in export\_graphviz(decision\_tree, out\_file, max\_depth, feature\_names, class\_names, label, filled, leaves\_parallel, impurity, node\_ids, proportion, rotate, rounded, special\_characters, precision, fontname)  
 869 out\_file = StringIO()  
 871 exporter = \_DOTTreeExporter(  
 872 out\_file=out\_file,  
 873 max\_depth=max\_depth,  
 (...)  
 886 fontname=fontname,  
 887 )  
--> 888 exporter.export(decision\_tree)  
 890 if return\_string:  
 891 return exporter.out\_file.getvalue()  
  
File /usr/local/lib/python3.9/site-packages/sklearn/tree/\_export.py:451, in \_DOTTreeExporter.export(self, decision\_tree)  
 449 if self.feature\_names is not None:  
 450 if len(self.feature\_names) != decision\_tree.n\_features\_in\_:  
--> 451 raise ValueError(  
 452 "Length of feature\_names, %d does not match number of features, %d"  
 453 % (len(self.feature\_names), decision\_tree.n\_features\_in\_)  
 454 )  
 455 # each part writes to out\_file  
 456 self.head()  
  
ValueError: Length of feature\_names, 4 does not match number of features, 5