



egaoharu_kensei 19 мар в 11:33

Бэггинг и случайный лес. Ключевые особенности и реализация с нуля на Python



Сложный



13 мин



2.5K

Python*, Data Mining*, Алгоритмы*, Машинное обучение*, Искусственный интеллект

Тutorial

Бэггинг и случайный лес

с нуля на Python 🐍



Далее пойдёт речь про бэггинг и мой самый любимый алгоритм — случайный лес. Не смотря на то, что это одни из самых первых алгоритмов среди семейства ансамблей, они до сих пор пользуются большой популярностью за счёт своей простоты и эффективности, зачастую не уступая бустингам в плане точности. О том, что это такое и как работает, далее в статье.

Ноутбук с данными алгоритмами можно загрузить на [Kaggle](#) (eng) и [GitHub](#) (rus).

Классификаторы и регрессоры с голосованием

Предположим, имеется несколько классификаторов, обученных на одних и тех же данных. Простым способом создания более сильного классификатора будет мажоритарное агрегирование прогнозов на их основе, то есть конечным прогнозом будет класс с наибольшим количеством голосов. Такой алгоритм называется *классификатором с жёстким голосованием (hard voting classifier)*.

Если же базовые классификаторы способны оценивать вероятности классов, то есть поддерживают метод «predict_proba», тогда можно создать классификатор, прогнозирующий класс с наибольшей вероятностью, усреднённый по всем базовым классификаторам. Такой алгоритм называется *классификатором с мягким голосованием (soft voting classifier)* и имеет более высокую точность за счёт придания большего веса голосам с высокой достоверностью.

В случае регрессии прогнозы базовых алгоритмов усредняются и такой алгоритм называется *регрессором с голосованием*.

В scikit-learn классификаторы и регрессоры с голосованием представлены в виде классов



+8



56



0

по отдельности, однако они по-прежнему плохо работают с выбросами и подвержены переобучению за счёт обучения каждой входящей в них модели на одинаковом наборе данных.

Бэггинг и вставка

Более продвинутый подход предусматривает многократное использование одной и той же базовой модели, обученной на разных случайных поднаборах тренировочного датасета. Если выборка осуществляется с возвращением, такой подход называется *бэггингом* (*bagging* — сокращение от *bootstrap aggregating*), а в случае выборки без замены — *вставкой* (*pasting*).

В статистике процедура генерирования повторной выборки с возвращением называется бутстрапированием (bootstrapping). Пример приведён ниже.

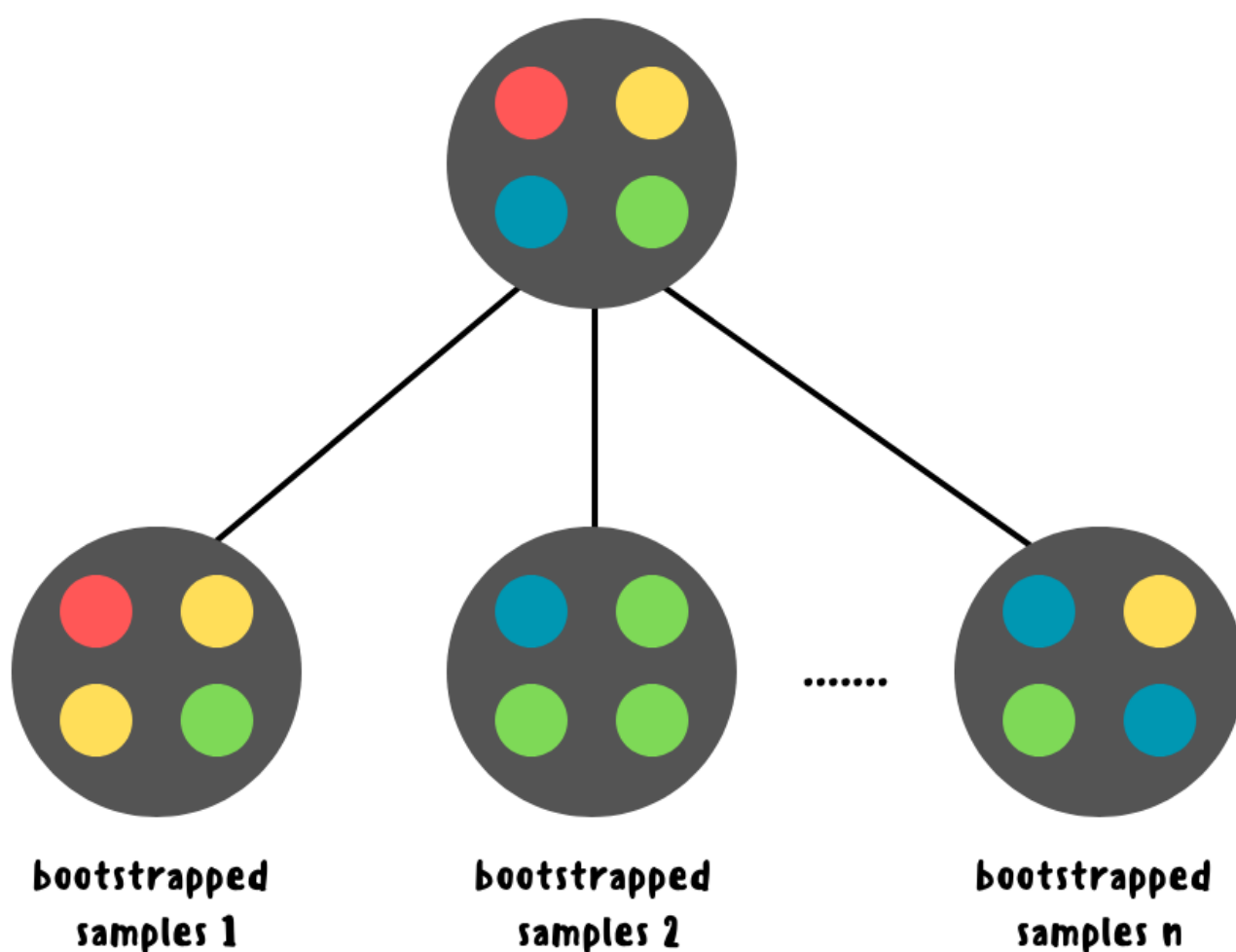


Схема бутстрапирования

После многократного обучения n одинаковых базовых моделей на бутстрапированных выборках, полученный ансамбль производит конечный прогноз, просто агрегируя прогнозы базовых моделей. В случае классификации функция агрегирования представляет собой статистическую моду, а в случае регрессии — среднее арифметическое. Такой подход позволяет уменьшить как смещение, так и дисперсию.

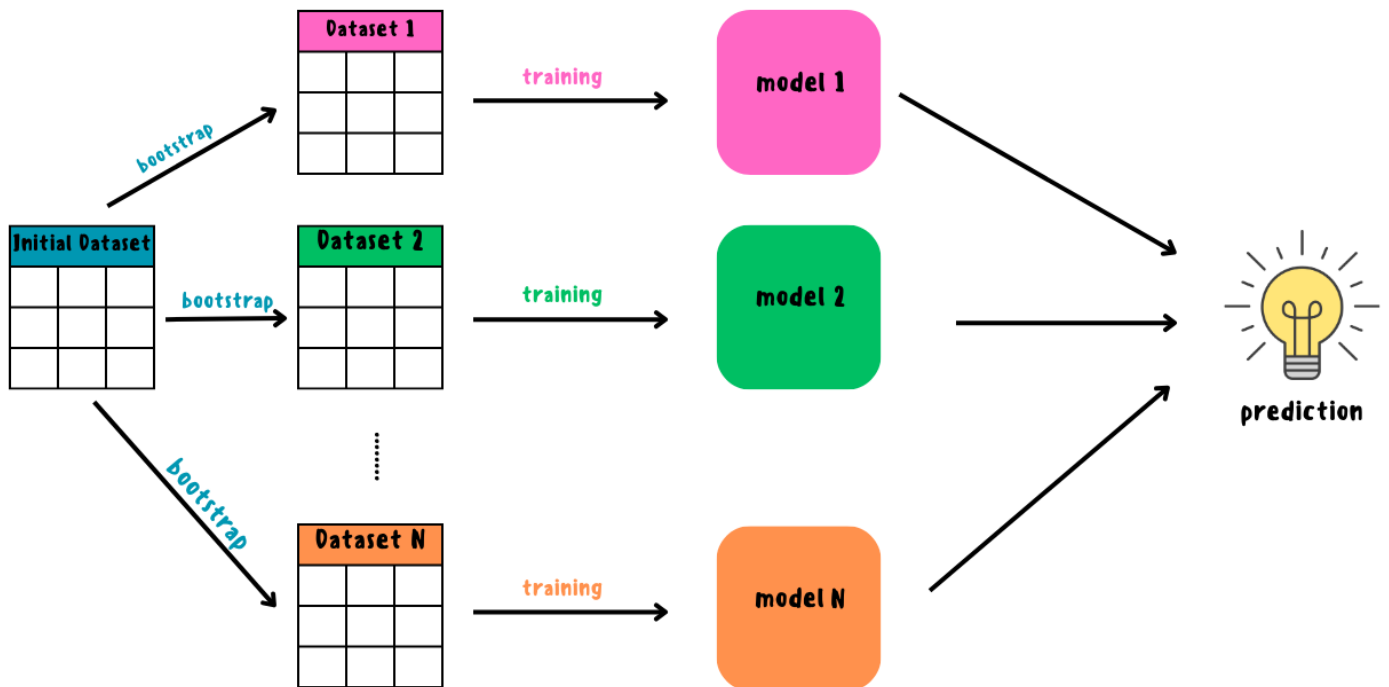


Схема работы бэггинга и вставки

Как можно было заметить, базовые модели могут обучаться и делать прогнозы параллельно, что делает бэггинг и вставку хорошо масштабируемыми методами — это одна из причин их широкого использования. В реализации `scikit-learn` для бэггинга *параллелизация реализована с помощью библиотеки `jllib`*, а количество используемых ядер устанавливается с помощью параметра «`n_jobs`» (при значении `-1` используются все доступные ядра). В нашей ручной реализации мы применим также параллельное обучение с использованием данной библиотеки.

В целом, бэггинг работает немного лучше вставки за счёт того, что бутстрапирование создаёт более отличные друг от друга обучающие выборки.

Оценка на неиспользуемых образцах

Учитывая, что при бэггинге выполняется выборка с возвращением, контролируемая параметром «`bootstrap=True`» (в случае `False` выполняется вставка), то в среднем из m обучающих образцов будет выбираться около 63% для каждой базовой модели. Оставшиеся 37% называются *неиспользуемыми обучающими образцами* (`out-of-bag` или `oob`), что можно легко доказать:

$$\lim_{m \rightarrow \infty} \left(1 - \frac{1}{m}\right)^m = \frac{1}{e} \approx 0.37$$

Стоит также отметить, что такие 37% процентов образцов будут разными для каждой базовой модели в ансамбле.

Учитывая, что базовые модели не используют `oob`-образцы во время обучения, их можно использовать в качестве отдельного тестового набора или при проведении кросс-валидации,

а выполнить оценку всего ансамбля можно путем усреднения oob-score каждой базовой модели, установив параметр «oob_score=True».

Методы случайных участков и случайных подпространств

Выбор случайным образом сразу обучающих образцов и признаков называется *методом случайных участков (random patches method)*, а выборка признаков (bootstrap_features=True и/или max_features < 1.0) с сохранением всех обучающих образцов называется *методом случайных подпространств (random subspaces method)*. Данные методы особенно полезны при использовании большого количества данных высокой размерности.

Стоит отметить, что выборка признаков позволяет создать более отличные друг от друга базовые модели, слегка увеличивая смещение и уменьшая дисперсию.

Случайный лес и особо случайные деревья

Случайный лес — это частный, оптимизированный случай бэггинга и вставки на основе решающих деревьев.

Алгоритм строится следующим образом:

1. изначально строится n бутстрапированных выборок, как правило, методом случайных подпространств;
2. n деревьев обучаются параллельно на полученных выборках и делают прогнозы на X_{test} ;
3. полученные прогнозы агрегируются модой в случае классификации и средним в случае регрессии.

Формулы для расчётов

Прогноз случайного леса для регрессии:

$$\hat{f}_{rf}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x)$$

Прогноз случайного леса для классификации:

$$\hat{C}_{rf}^B(x) = \text{majority vote}[\hat{C}_b(x)]_1^B$$

Прогноз b -го бутстрапированного дерева в случае регрессии: $T_b(x)$

Спрогнозированный класс b -го бутстрапированного дерева в случае классификации: $\hat{C}_b(x)$

Максимальное число признаков для регрессии: $m = \frac{p}{3}$

Максимальное число признаков для регрессии в реализации scikit-learn: $m = p$

Максимальное число признаков для классификации: $m = \sqrt{p}$

Число всех признаков в датасете: p

Деревья в случайном лесе можно сделать ещё более случайными за счёт выбора случайных пороговых значений для каждого признака вместо поиска наилучшего, как это реализовано в CART. Лес с такими случайными деревьями называется **ансамблем особо случайных деревьев (extremely randomized trees ensemble или extra-trees)**. В особо случайных деревьях каждое дерево ещё более отлично друг от друга, чем в случайном лесе, что также увеличивает смещение и снижает дисперсию. Помимо этого, такие деревья обучаются гораздо быстрее.

Значимость признаков

Ещё одним полезным свойством случайного леса является измерение относительной значимости каждого признака, основанное на том, как сильно в узлах дерева снижается загрязнённость в среднем с использованием текущего признака. Проще говоря, значимость признака является взвешенным средним, где вес каждого узла соответствует количеству ассоциированных с ним образцов: чем больше вес признака, тем больше его вклад в общую точность модели.

Стоит отметить, что в машинном обучении существуют более продвинутые методы интерпретации результатов модели, например SHAP или LIME.

Импорт необходимых библиотек

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_absolute_percentage_error
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import BaggingClassifier, VotingClassifier, ExtraTreesClassifier
```

```
from joblib import Parallel, delayed
from mlxtend.plotting import plot_decision_regions
```

Реализация на Python с нуля

```
class RandomForest:
    def __init__(self, regression=False, n_estimators=100, max_depth=None,
                 max_features=1.0, n_jobs=-1, random_state=0, ccp_alpha=0.0):
        self.regression = regression
        self.n_estimators = n_estimators
        self.max_depth = max_depth
        self.max_features = max_features
        self.n_jobs = n_jobs
        self.random_state = random_state
        self.ccp_alpha = ccp_alpha
        self.trained_trees_info = []
        self.general_random = np.random.RandomState(self.random_state)

    #bootstrapping with random subspaces method
    def _rsm_bootstrapping(self, X, y):
        n_samples, n_features = X.shape
        if self.regression:
            max_features = self.max_features * n_features
        else:
            max_features = np.sqrt(n_features)

        sample_indexes = self.general_random.choice(n_samples, n_samples)
        features = self.general_random.choice(X.columns, round(max_features))
        X_b, y_b = X.iloc[sample_indexes][features], y.iloc[sample_indexes]

        return X_b, y_b

    def _train_tree(self, X, y):
        if self.regression:
            tree = DecisionTreeRegressor(max_depth=self.max_depth,
                                         random_state=self.random_state,
                                         ccp_alpha=self.ccp_alpha)
        else:
            tree = DecisionTreeClassifier(max_depth=self.max_depth,
                                         random_state=self.random_state,
                                         ccp_alpha=self.ccp_alpha)

        return tree.fit(X, y), X.columns

    def fit(self, X, y):
        boot_data = (self._rsm_bootstrapping(X, y) for _ in range(self.n_estimators))
        train_trees = (delayed(self._train_tree)(X_b, y_b) for X_b, y_b in boot_data)
```

```

self.trained_trees_info = Parallel(n_jobs=self.n_jobs)(train_trees)

def predict(self, samples):
    prediction = (delayed(tree_i.predict)(samples[tree_i_features])
                  for (tree_i, tree_i_features) in self.trained_trees_info)

    trees_predictions = pd.DataFrame(Parallel(n_jobs=self.n_jobs)(prediction))

    if self.regression:
        forest_prediction = trees_predictions.mean(axis=0)
    else:
        forest_prediction = trees_predictions.mode(axis=0).iloc[0]

    return np.array(forest_prediction)

```

Код для отрисовки графика

```

def decision_boundary_plot(X, y, X_train, y_train, clf, feature_indexes, title=None):
    feature1_name, feature2_name = X.columns[feature_indexes]
    X_feature_columns = X.values[:, feature_indexes]
    X_train_feature_columns = X_train.values[:, feature_indexes]
    clf.fit(X_train_feature_columns, y_train.values)

    plot_decision_regions(X=X_feature_columns, y=y.values, clf=clf)
    plt.xlabel(feature1_name)
    plt.ylabel(feature2_name)
    plt.title(title)

```

Загрузка датасетов

Для обучения моделей будет использован Glass Classification датасет, где необходимо верно определить тип стекла по его признакам. В случае регрессии используется Diabetes датасет из scikit-learn.

```

df_path = "/content/drive/MyDrive/glass.csv"
glass_df = pd.read_csv(df_path)
X1, y1 = glass_df.iloc[:, :-1], glass_df.iloc[:, -1]
y1 = pd.Series(LabelEncoder().fit_transform(y1))
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, random_state=0)
print(glass_df)

```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1

2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
..
209	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

[214 rows x 10 columns]

```
X2, y2 = load_diabetes(return_X_y=True, as_frame=True)
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, random_state=0)
print(X2, y2, sep='\n')
```

	age	sex	bmi	bp	s1	s2	s3 \
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142
..
437	0.041708	0.050680	0.019662	0.059744	-0.005697	-0.002566	-0.028674
438	-0.005515	0.050680	-0.015906	-0.067642	0.049341	0.079165	-0.028674
439	0.041708	0.050680	-0.015906	0.017293	-0.037344	-0.013840	-0.024993
440	-0.045472	-0.044642	0.039062	0.001215	0.016318	0.015283	-0.028674
441	-0.045472	-0.044642	-0.073030	-0.081413	0.083740	0.027809	0.173816

	s4	s5	s6
0	-0.002592	0.019907	-0.017646
1	-0.039493	-0.068332	-0.092204
2	-0.002592	0.002861	-0.025930
3	0.034309	0.022688	-0.009362
4	-0.002592	-0.031988	-0.046641
..
437	-0.002592	0.031193	0.007207
438	0.034309	-0.018114	0.044485
439	-0.011080	-0.046883	0.015491
440	0.026560	0.044529	-0.025930
441	-0.039493	-0.004222	0.003064

[442 rows x 10 columns]

0	151.0
1	75.0
2	141.0
3	206.0


```
4      135.0
...
437    178.0
438    104.0
439    132.0
440    220.0
441     57.0
```

```
Name: target, Length: 442, dtype: float64
```

Обучение моделей и оценка полученных результатов

Как и ожидалось, бэггинг и случайный лес показали прирост в точности по сравнению с деревом решений. Разница ещё более заметна с использованием особо случайных деревьев, а вот классификатор с голосованием прироста в точности не дал.

Стоит также отметить, что особо случайные деревья не всегда выигрывают в плане точности в сравнении со случайным лесом и единственным достоверным способом узнать какой из алгоритмов лучше, является их прямое сравнение с подбором разных гиперпараметров.

Ниже приведены полученные результаты и графики решающих границ для каждого алгоритма.

Random Forest для классификации

```
random_forest_classifier = RandomForest(random_state=0)
random_forest_classifier.fit(X1_train, y1_train)
rf_clf_pred_res = random_forest_classifier.predict(X1_test)
rf_clf_accuracy = accuracy_score(rf_clf_pred_res, y1_test)
print(rf_clf_pred_res)
print(f'rf_clf accuracy: {rf_clf_accuracy}')
```



```
sk_random_forest_classifier = RandomForestClassifier(random_state=0)
sk_random_forest_classifier.fit(X1_train, y1_train)
sk_rf_clf_pred_res = sk_random_forest_classifier.predict(X1_test)
sk_rf_clf_accuracy = accuracy_score(sk_rf_clf_pred_res, y1_test)
print(sk_rf_clf_pred_res)
print(f'sk_rf_clf accuracy: {sk_rf_clf_accuracy}')
```



```
[5 0 1 4 3 1 0 1 1 1 0 0 1 1 1 5 0 1 2 0 1 0 5 5 0 0 5 0 1 1 0 0 1 0 0 0 0
 0 0 5 1 4 1 0 1 1 0 1 0 1 0 1 5 0]
rf_clf accuracy: 0.7222222222222222
```



```
[5 0 1 4 3 1 0 1 1 1 1 0 1 1 1 5 2 1 2 1 3 0 5 5 0 0 5 0 1 1 0 0 1 0 0 0 0
 0 0 5 3 4 1 0 1 1 0 1 0 1 0 1 5 0]
sk_rf_clf accuracy: 0.7037037037037037
```

Random Forest для регрессии

```
random_forest_regressor = RandomForest(regression=True, random_state=0)
random_forest_regressor.fit(X2_train, y2_train)
rf_reg_pred_res = random_forest_regressor.predict(X2_test)
mape = mean_absolute_percentage_error(rf_reg_pred_res, y2_test)
print(f'mape: {mape}')
print(rf_reg_pred_res)
```

```
sk_random_forest_regressor = RandomForestRegressor(random_state=0)
sk_random_forest_regressor.fit(X2_train, y2_train)
sk_rf_reg_pred_res = sk_random_forest_regressor.predict(X2_test)
sk_mape = mean_absolute_percentage_error(sk_rf_reg_pred_res, y2_test)
print(f'sk_mape: {sk_mape}')
print(sk_rf_reg_pred_res)
```

mape: 0.3162321058884571

```
[233.89 236.14 160.42 114.39 182.3  233.59 101.64 231.75 134.16 209.1
 157.81 157.92 148.37 112.27 276.42 106.68 148.94  82.27 110.39 221.53
 163.31 132.82 158.98 114.09 201.66 187.82 144.77  81.82 194.05 173.86
 174.28  84.67 135.73 162.31 148.59 188.1  163.34 144.86 108.44 212.66
 111.    159.73 123.71 173.85 177.94  87.07 122.24 141.52 111.05 220.33
 159.28  75.79 156.81 173.54 234.88 204.58 180.04 118.67 146.71 167.36
 216.98 153.32 132.07 105.45 239.33 150.75  93.58 224.83 200.27 112.72
  86.13 135.73 117.98 135.13 136.06 180.59 137.36 225.1  244.43 209.96
 133.33 210.47  78.8  240.79 124.75  96.83 144.62 189.53 112.04 146.64
 103.61 114.46  95.39 186.43 105.3   90.85 232.33 216.32 143.46 156.55
 156.59 112.41 176.95 100.51 227.5  143.1  213.6  255.85 106.35  92.22
 237.11]
```

sk_mape: 0.3313513936725928

```
[250.61 251.57 169.22  92.96 208.8  261.45  91.43 240.08 128.06 240.07
 167.75 151.67 127.6   95.09 289.77  90.21 162.    77.62 107.74 220.32
 186.06 125.12 163.62 130.02 218.6  189.48 130.9   81.49 210.22 156.14
 180.66  74.    110.35 161.2  143.94 183.56 152.95 124.07  95.37 230.31
 100.23 166.91 125.19 185.65 184.5   75.85 115.77 137.16 110.79 249.37
 149.82  71.63 178.53 165.67 262.99 190.86 186.88  99.41 141.24 168.4
 252.62 155.37 126.8   99.8  261.6  151.19  81.    225.57 221.44 118.12
  83.09 148.36 115.91 128.14 130.36 178.47 115.51 229.29 262.53 196.85
 114.72 205.47  79.18 246.56 119.8   88.36 149.19 211.39 100.69 142.92
 101.3   98.92  96.19 171.63 101.3   94.    253.01 237.53 138.73 146.03
 143.91 107.73 167.38 101.56 245.    147.27 231.09 263.92 102.98  89.12
 260.02]
```

```

importances = sk_random_forest_regressor.feature_importances_
feature_importances = pd.Series(importances, index=X2_train.columns)
feature_importances.sort_values(ascending=True).plot(kind='barh')

```

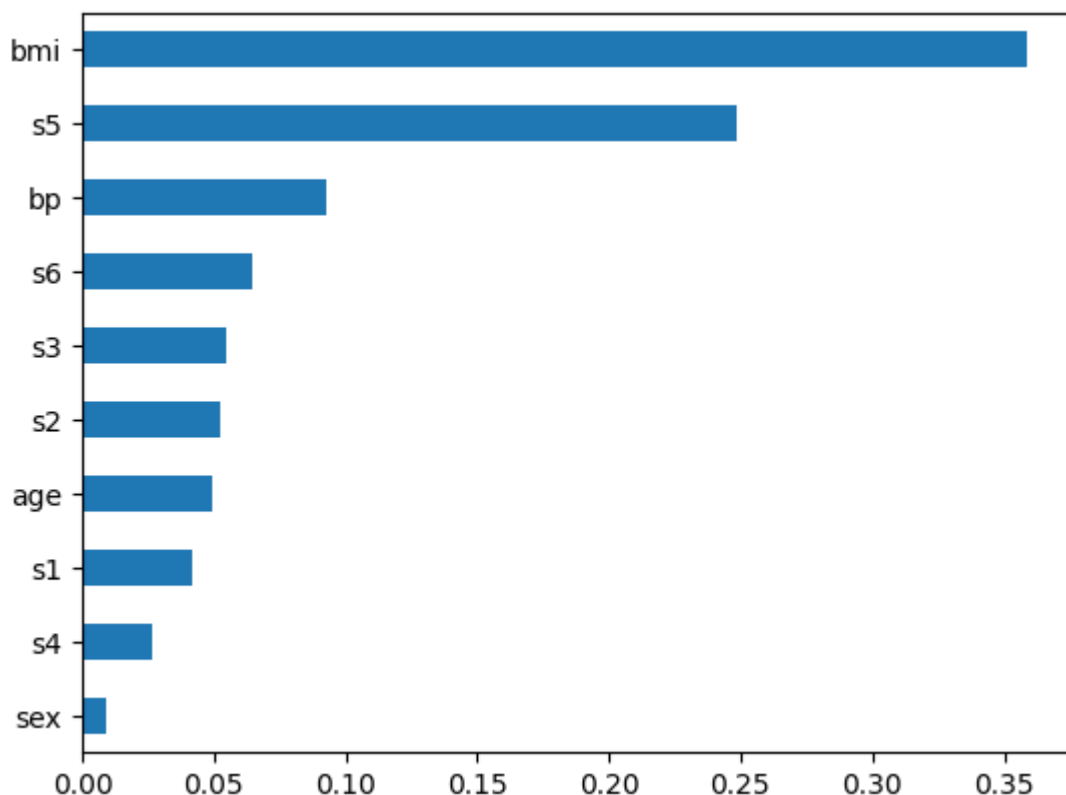


График важности признаков в Random Forest

DecisionTreeClassifier

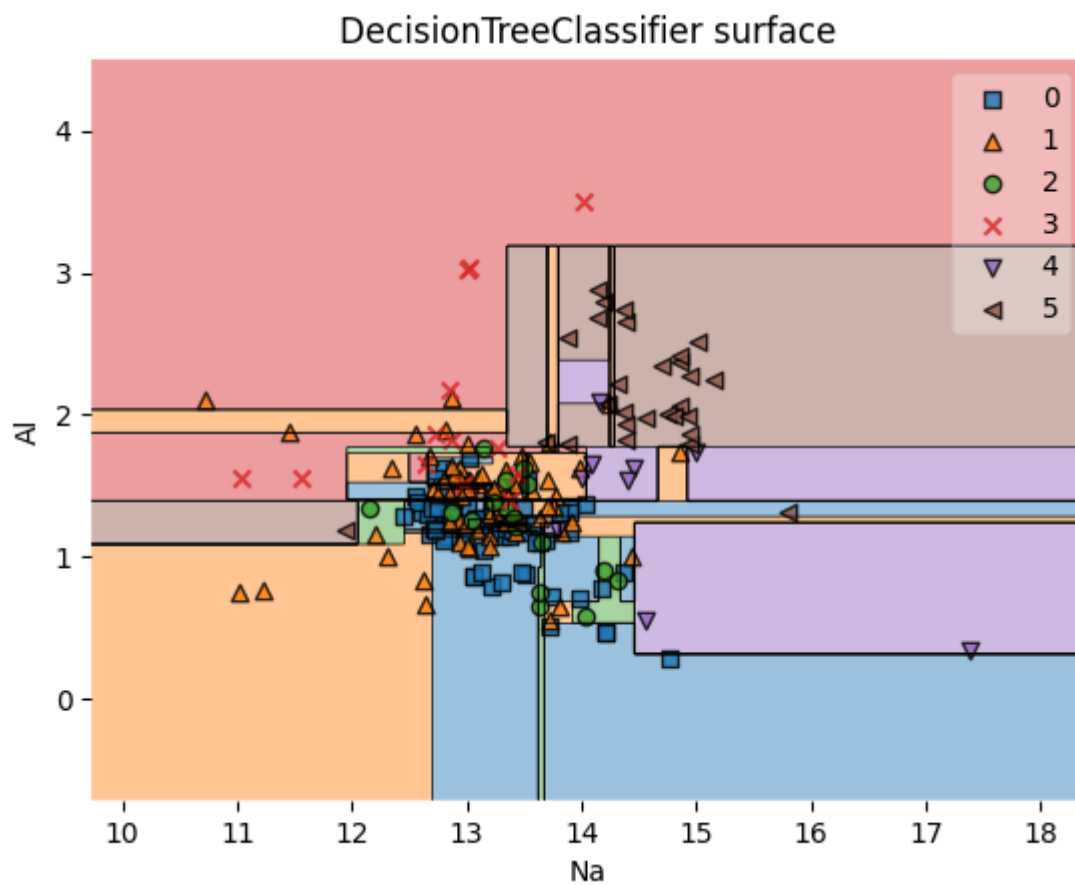
```

sk_tree_clf = DecisionTreeClassifier(random_state=0)
sk_tree_clf.fit(X1_train, y1_train)
tree_pred_res = sk_tree_clf.predict(X1_test)
tree_accuracy = accuracy_score(tree_pred_res, y1_test)
print(f'tree accuracy: {tree_accuracy}')
print(tree_pred_res)

feature_indexes = [1, 3]
title1 = 'DecisionTreeClassifier surface'
decision_boundary_plot(X1, y1, X1_train, y1_train, sk_tree_clf, feature_indexes, title1)

tree accuracy: 0.6111111111111112
[5 0 1 1 3 1 0 1 1 1 1 2 1 1 1 5 2 1 0 1 3 0 5 3 0 0 5 0 0 1 0 0 1 2 0 0 0
 2 0 5 3 4 1 0 1 1 0 1 0 1 0 4 5 2]

```



VotingClassifier

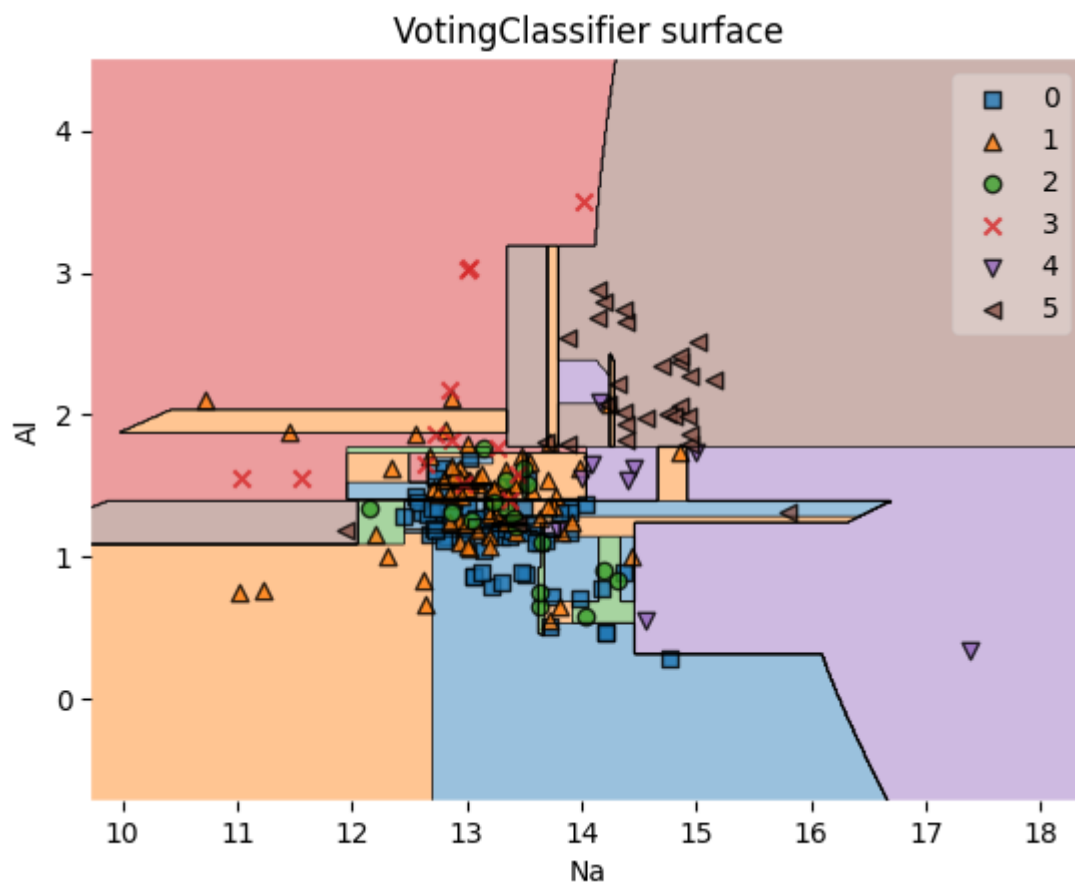
```

estimators = [('lr', LogisticRegression(random_state=0, max_iter=10000)),
              ('dt', DecisionTreeClassifier(random_state=0)),
              ('svc', SVC(probability=True, random_state=0))]

voting_clf = VotingClassifier(estimators, voting='soft')
voting_clf.fit(X1_train, y1_train)
voting_clf_pred_res = voting_clf.predict(X1_test)
voting_clf_accuracy = accuracy_score(voting_clf_pred_res, y1_test)
print(f'voting_clf accuracy: {voting_clf_accuracy}')
print(voting_clf_pred_res)

feature_indexes = [1, 3]
title2 = 'VotingClassifier surface'
decision_boundary_plot(X1, y1, X1_train, y1_train, voting_clf, feature_indexes, title2)

voting_clf accuracy: 0.6111111111111112
[5 0 1 1 3 1 0 1 1 1 1 2 1 1 1 5 2 1 0 1 3 0 5 3 0 0 5 0 0 1 0 0 1 2 0 0 0
 2 0 5 3 4 1 0 1 1 0 1 0 1 0 4 5 2]
```



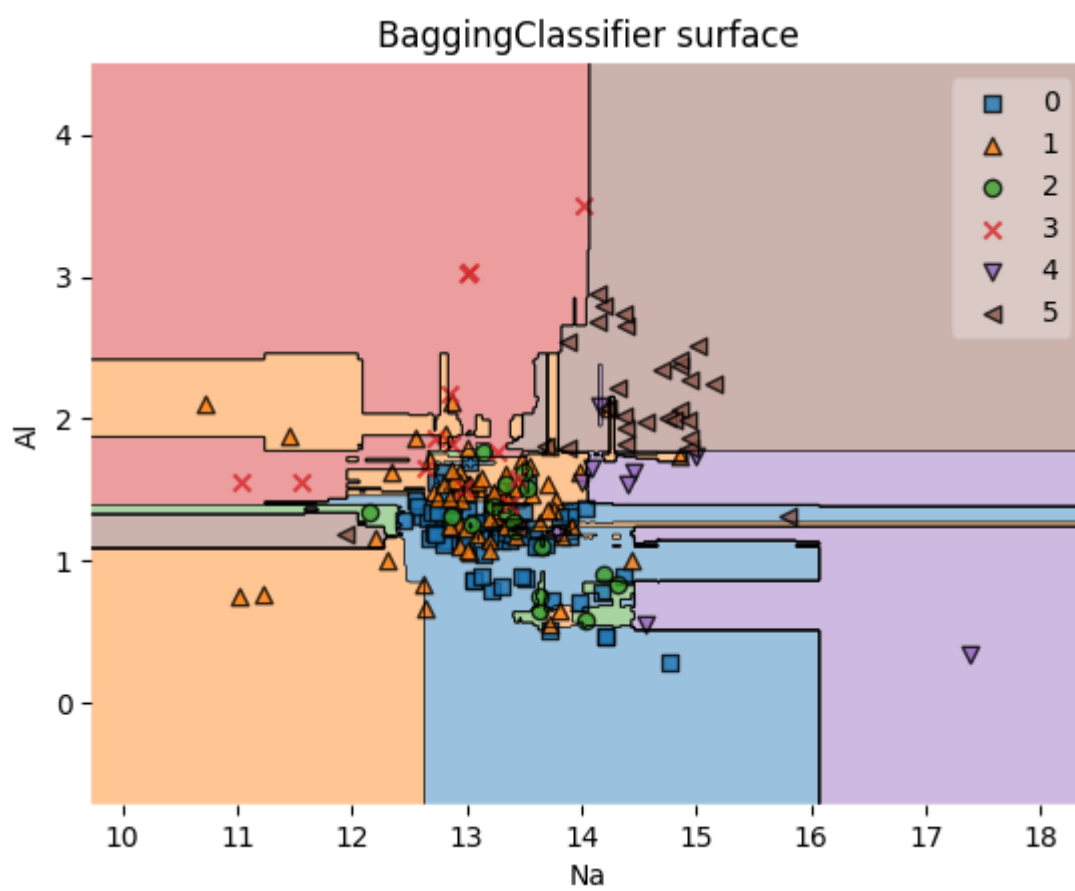
BaggingClassifier

```
dt_clf = DecisionTreeClassifier(random_state=0)
bagging_clf = BaggingClassifier(estimator=dt_clf, n_estimators=100, random_state=0)
bagging_clf.fit(X1_train, y1_train)
bagging_pred_res = bagging_clf.predict(X1_test)
bagging_clf_accuracy = accuracy_score(bagging_pred_res, y1_test)
print(f'bagging_clf accuracy: {bagging_clf_accuracy}')
print(bagging_pred_res)

feature_indexes = [1, 3]
title3 = 'BaggingClassifier surface'
decision_boundary_plot(X1, y1, X1_train, y1_train, bagging_clf, feature_indexes, title3)
```

bagging_clf accuracy: 0.6851851851851852

```
[5 0 1 4 3 1 0 1 1 1 1 0 1 1 1 5 2 1 0 1 3 0 5 5 0 0 5 0 1 1 0 0 1 0 1 0 0
 0 0 5 5 4 1 0 1 1 0 1 0 1 0 4 5 0]
```

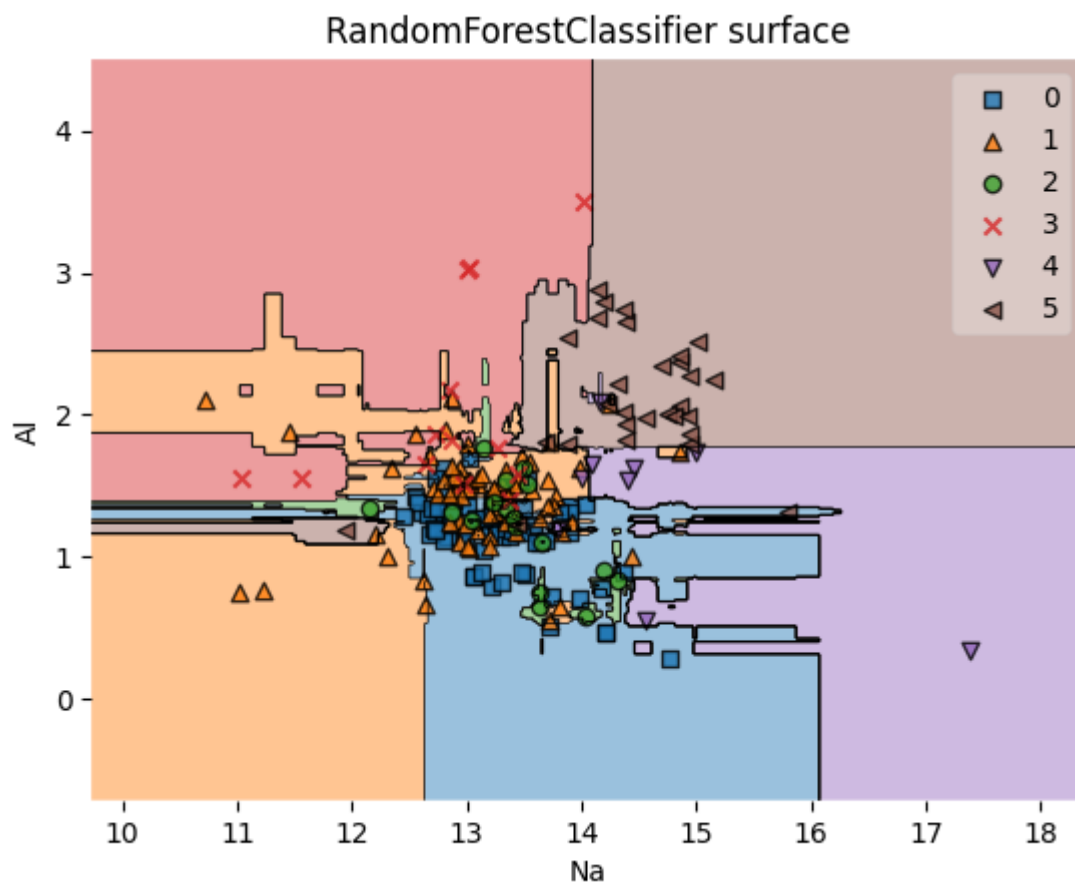


RandomForestClassifier

```
sk_rf_clf = sk_random_forest_classifier  
print(sk_rf_clf_accuracy)
```

```
feature_indexes = [1, 3]  
title4 = 'RandomForestClassifier surface'  
decision_boundary_plot(X1, y1, X1_train, y1_train, sk_rf_clf, feature_indexes, title4)
```

0.7037037037037037



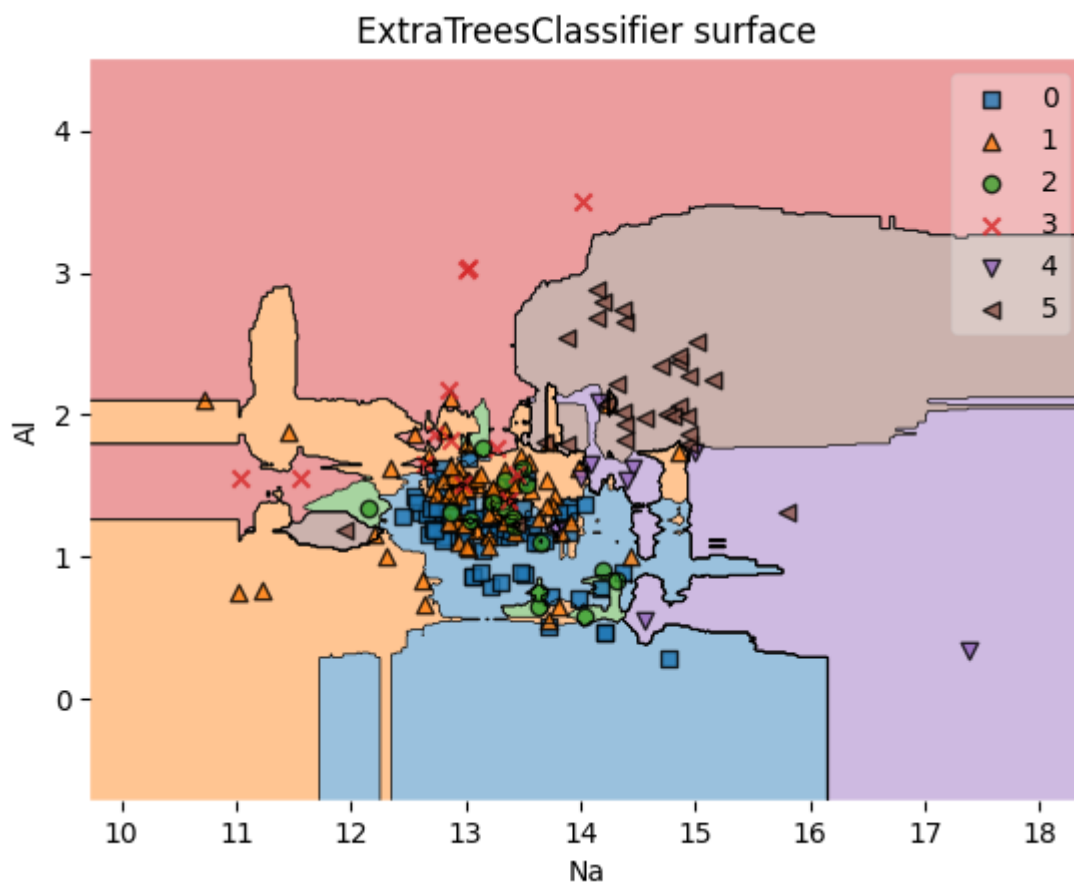
ExtraTreesClassifier

```
extra_trees_clf = ExtraTreesClassifier(random_state=0)
extra_trees_clf.fit(X1_train, y1_train)
et_clf_pred_res = extra_trees_clf.predict(X1_test)
et_clf_accuracy = accuracy_score(et_clf_pred_res, y1_test)
print(f'et_clf accuracy: {et_clf_accuracy}')
print(et_clf_pred_res)

feature_indexes = [1, 3]
title5 = 'ExtraTreesClassifier surface'
decision_boundary_plot(X1, y1, X1_train, y1_train, extra_trees_clf, feature_indexes, title5)
```

et_clf accuracy: 0.7222222222222222

```
[5 0 1 4 3 1 0 1 1 1 0 1 1 1 5 0 1 2 1 3 0 5 5 0 0 5 0 1 1 0 0 1 0 0 0 0
 0 0 5 1 4 1 0 1 1 0 1 0 1 0 1 5 0]
```



Преимущества и недостатки случайного леса

Преимущества:

- высокая точность прогнозов, часто сравнимая с бустингами;
- устойчивость к выбросам и несбалансированным датасетам;
- высокая скорость работы и хорошая масштабируемость;
- возможность проведения кластеризации;
- хорошо работает с пропусками;
- часто устойчив к переобучению.

Недостатки:

- более сложная интерпретируемость результатов в сравнении с деревом решений;
- склонность к переобучению на сильно зашумленных данных;
- предвзятость к категориальным признакам с большим количеством уникальных классов: деревья будут сильнее подстраиваться под эти признаки из-за более высокого информационного прироста.

Дополнительные источники

Статья «RANDOM FORESTS», Leo Breiman.

Документация:

- описание ансамблей;
- VotingClassifier;
- VotingRegressor;
- BaggingClassifier;
- BaggingRegressor;
- RandomForestClassifier;
- RandomForestRegressor;
- ExtraTreesClassifier;
- ExtraTreesRegressor.

Лекции:

- один ;
- два.

Пошаговое построение Random Forest:

- один;
 - два;
 - три.
-

◀ **Дерево решений (CART) | Алгоритмы AdaBoost** ▶

Теги: бэггинг, вставка, случайный лес, bagging, random forest, реализация с нуля, python, алгоритмы машинного обучения, data science, машинное обучение

Хэбы: Python, Data Mining, Алгоритмы, Машинное обучение, Искусственный интеллект



27

Карма

91

Рейтинг

Егор Захаренко @egaoharu_kensei

Пользователь

Подписан



Комментировать

Публикации

ЛУЧШИЕ ЗА СУТКИ

ПОХОЖИЕ



Grigory_Otrepyev 5 часов назад

Российская микроэлектроника — два года спустя

Сложный

9 мин

13K

Дайджест

+140

42

58 +58



bodyawm 6 часов назад

В моих жилах течет моддерская кровь: как и зачем я променял оригинальный айфон на нерабочую подделку за 1500 рублей?

Средний

18 мин

2.9K

Обзор

+34

19

22 +22



HallEffect вчера в 17:13

Ферма тестирования SberDevices

Средний

14 мин

3.7K

Обзор

+34

26

4 +4



obondar 5 часов назад

От хаоса к порядку. Как мы внедряем стандарты в CDEK

Средний

11 мин

817

Кейс

+31

40

6 +6



wofs 7 часов назад

Как мы сделали Embedded Controller для ПЛК на Linux



Средний



15 мин



2.5K

Кейс



+28



16



14 +14



Lunathecat 5 часов назад

Паяем классическую педаль Marshall Bluesbreaker



Простой



8 мин



1K

Ретроспектива



+27



10



0



vickylikh 8 часов назад

Будущее Kubernetes и DevOps: строим прогнозы на 10 лет



Простой



13 мин



3.5K

Аналитика



+22



15



5 +5



zakhmatov 7 часов назад

Как DDoS-атаки стали для нас рутиной и как ML помогает их отражать



10 мин



955



+19



10



0



rananyev 23 часа назад

Sub-GHz во Flipper Zero и бесконечное множество внешних антенн



Простой



6 мин



4.2K

Обзор



+19



33



17 +17



vasilisa_b 7 часов назад

Как в России в XIX веке компьютер изобрели

Всему учён и изловчён: где взять знания, которые в работе точно пригодятся

Турбо

Показать еще

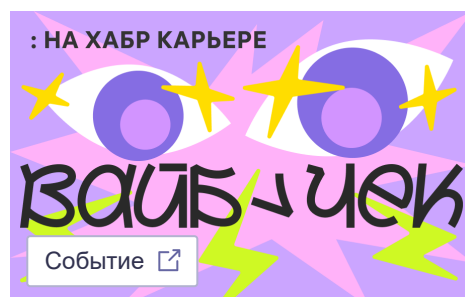
МИНУТОЧКУ ВНИМАНИЯ



Как бессонница в час ночной,
меняет промокодище облик твой





Где достать знания, которые в
работе точно пригодятся





Выбирайте команду тестирования
по вайбам


КУРСЫ

 **Администрирование PostgreSQL 13. Настройка и мониторинг**
25 марта 2024 · Hi-TECH Academy

 **Продвинутый SQL**
25 марта 2024 · Нетология

 **SQL и получение данных**
25 марта 2024 · Нетология

 **Django: создание backend-приложений**
25 марта 2024 · Нетология

 **Аналитик данных: расширенный курс**
25 марта 2024 · Нетология

Больше курсов на Хабр Карьере

ЧИТАЮТ СЕЙЧАС

Учёные назвали ударом для российской школы физики элементарных частиц последствия прекращения сотрудничества с ЦЕРН

13K 52 +52

Российская микроэлектроника — два года спустя

13K 58 +58

Разработчик с помощью дипфейка в реальном времени прошёл собеседование за друга

10K 66 +66

Очередное пособие по рынку труда, или где же вы 300к находите. Март 2024

41K 188 +188

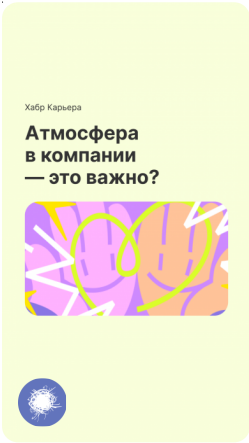
Десятки ведущих учёных подписали документ, направленный на предотвращение разработки биологического оружия при помощи ИИ

30K 41 +41

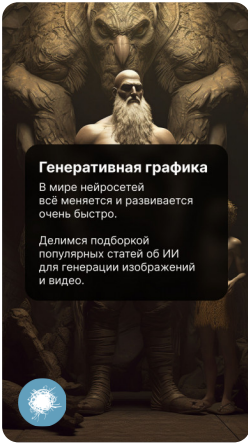
Всему учён и изловчён: где взять знания, которые в работе точно пригодятся

Турбо

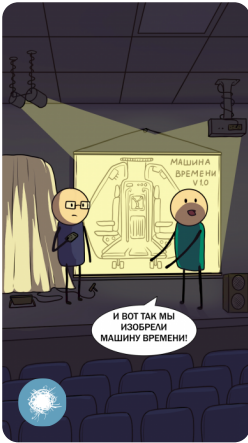
ИСТОРИИ



Вайб-чек для тестировщиков



Нейросети: интересное



Как продвинуть машину времени?



Наука сна

РАБОТА

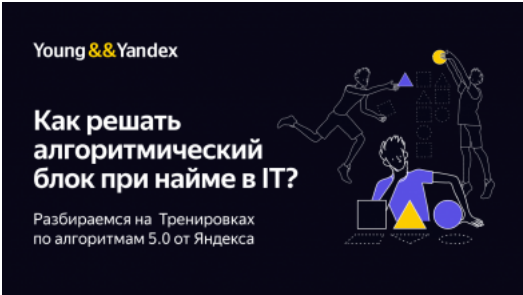
Python разработчик
127 вакансий

Django разработчик
38 вакансий

Data Scientist
61 вакансия

Все вакансии

БЛИЖАЙШИЕ СОБЫТИЯ



Серия занятий
«Тренировки по алгоритмам 5.0» от Яндекса

1 марта – 19 апреля
19:00
Онлайн

[Подробнее в календаре](#)



Тестировщики,
выбирайте себе команду по вайбам на Хабр Карьере

18 – 24 марта
09:00 – 23:00
Онлайн

[Подробнее в календаре](#)



«GoCloud 2024. Ог
рани будущего» - конференция Clo
у про облака

21 марта 09:00
Москва • Онлайн

[Подробнее в календаре](#)

Ваш аккаунт

- Профиль
- Трекер
- Диалоги
- Настройки
- ППА

Разделы

- Статьи
- Новости
- Хабы
- Компании
- Авторы
- Песочница

Информация

- Устройство сайта
- Для авторов
- Для компаний
- Документы
- Соглашение
- Конфиденциальность

Услуги

- Корпоративный блог
- Медийная реклама
- Нативные проекты
- Образовательные программы
- Стартапам



[Настройка языка](#)

[Техническая поддержка](#)

© 2006–2024, Habr