

# Ансамбль деревьев

В данном ноутбуке рассмотрим работу бэггинга над решающими деревьями, случайного леса и линейной комбинации разных алгоритмов.

Для начала загрузим данные и посмотрим на них.

```
In [ ]: from sklearn import datasets
        from sklearn.model_selection import train_test_split
        import numpy as np

        ds = datasets.load_diabetes()
```

```
In [ ]: ds.feature_names
```

```
Out[ ]: ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

```
In [ ]: print(ds['DESCR'])
```

```
.. _diabetes_dataset:
```

Diabetes dataset

-----

Ten baseline variables, age, sex, body mass index, average blood pressure, and six blood serum measurements were obtained for each of  $n = 442$  diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

**\*\*Data Set Characteristics:\*\***

:Number of Instances: 442

:Number of Attributes: First 10 columns are numeric predictive values

:Target: Column 11 is a quantitative measure of disease progression one year after baseline

:Attribute Information:

- Age
- Sex
- Body mass index
- Average blood pressure
- S1
- S2
- S3
- S4
- S5
- S6

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times  $\sqrt{n_{\text{samples}}}$  (i.e. the sum of squares of each column totals 1).

Source URL:

<https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html>

For more information see:

Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," *Annals of Statistics* (with discussion), 407-499.

([https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle\\_2002.pdf](https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf))

```
In [ ]: print (ds.target)
```

```
[151.  75. 141. 206. 135.  97. 138.  63. 110. 310. 101.  69. 179. 185.
118. 171. 166. 144.  97. 168.  68.  49.  68. 245. 184. 202. 137.  85.
131. 283. 129.  59. 341.  87.  65. 102. 265. 276. 252.  90. 100.  55.
 61.  92. 259.  53. 190. 142.  75. 142. 155. 225.  59. 104. 182. 128.
 52.  37. 170. 170.  61. 144.  52. 128.  71. 163. 150.  97. 160. 178.
 48. 270. 202. 111.  85.  42. 170. 200. 252. 113. 143.  51.  52. 210.
 65. 141.  55. 134.  42. 111.  98. 164.  48.  96.  90. 162. 150. 279.
 92.  83. 128. 102. 302. 198.  95.  53. 134. 144. 232.  81. 104.  59.
246. 297. 258. 229. 275. 281. 179. 200. 200. 173. 180.  84. 121. 161.
 99. 109. 115. 268. 274. 158. 107.  83. 103. 272.  85. 280. 336. 281.
118. 317. 235.  60. 174. 259. 178. 128.  96. 126. 288.  88. 292.  71.
197. 186.  25.  84.  96. 195.  53. 217. 172. 131. 214.  59.  70. 220.
268. 152.  47.  74. 295. 101. 151. 127. 237. 225.  81. 151. 107.  64.
138. 185. 265. 101. 137. 143. 141.  79. 292. 178.  91. 116.  86. 122.
 72. 129. 142.  90. 158.  39. 196. 222. 277.  99. 196. 202. 155.  77.
191.  70.  73.  49.  65. 263. 248. 296. 214. 185.  78.  93. 252. 150.
 77. 208.  77. 108. 160.  53. 220. 154. 259.  90. 246. 124.  67.  72.
257. 262. 275. 177.  71.  47. 187. 125.  78.  51. 258. 215. 303. 243.
 91. 150. 310. 153. 346.  63.  89.  50.  39. 103. 308. 116. 145.  74.
 45. 115. 264.  87. 202. 127. 182. 241.  66.  94. 283.  64. 102. 200.
265.  94. 230. 181. 156. 233.  60. 219.  80.  68. 332. 248.  84. 200.
 55.  85.  89.  31. 129.  83. 275.  65. 198. 236. 253. 124.  44. 172.
114. 142. 109. 180. 144. 163. 147.  97. 220. 190. 109. 191. 122. 230.
242. 248. 249. 192. 131. 237.  78. 135. 244. 199. 270. 164.  72.  96.
306.  91. 214.  95. 216. 263. 178. 113. 200. 139. 139.  88. 148.  88.
243.  71.  77. 109. 272.  60.  54. 221.  90. 311. 281. 182. 321.  58.
262. 206. 233. 242. 123. 167.  63. 197.  71. 168. 140. 217. 121. 235.
245.  40.  52. 104. 132.  88.  69. 219.  72. 201. 110.  51. 277.  63.
118.  69. 273. 258.  43. 198. 242. 232. 175.  93. 168. 275. 293. 281.
 72. 140. 189. 181. 209. 136. 261. 113. 131. 174. 257.  55.  84.  42.
146. 212. 233.  91. 111. 152. 120.  67. 310.  94. 183.  66. 173.  72.
 49.  64.  48. 178. 104. 132. 220.  57.]
```

Разделим данные на обучающую и тестовую часть.

```
In [ ]: X = ds.data
Y = ds.target
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.5, test_size=
0.5)
```

## 1. Решающее дерево

Построим одно решающее дерево по обучающей выборке. Будем использовать гиперпараметры по умолчанию. Это значит, что мы никак не будем использовать регуляризацию для дерева (ограничивать глубину и число листьев).

```
In [ ]: from sklearn.tree import DecisionTreeRegressor
```

```
In [ ]: clf = DecisionTreeRegressor()
```

```
In [ ]: clf.fit(X_train, y_train)

Out[ ]: DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                             max_leaf_nodes=None, min_impurity_decrease=0.0,
                             min_impurity_split=None, min_samples_leaf=1,
                             min_samples_split=2, min_weight_fraction_leaf=0.0,
                             presort=False, random_state=None, splitter='best')

In [ ]: from sklearn.metrics import mean_squared_error

In [ ]: mean_squared_error(y_test, clf.predict(X_test))

Out[ ]: 6618.27149321267

In [ ]: mean_squared_error(y_train, clf.predict(X_train))

Out[ ]: 0.0
```

MSE на обучающей выборке равно нулю. Построенное дерево переобучилось и не ошиблось ни на одном объекте из обучения.

## 2. Бэггинг над решающими деревьями

```
In [ ]: res = np.zeros(X_test.shape[0])
        N = 100
        clf = DecisionTreeRegressor()
        for i in range(N):
            ind = np.random.choice(np.arange(X_train.shape[0]),
                                   size = X_train.shape[0], replace = True)
            clf.fit(X_train[ind], y_train[ind])
            res += clf.predict(X_test)

In [ ]: mean_squared_error(y_test, res/N)

Out[ ]: 3805.171399095022
```

Как известно из лекций решающее дерево без регуляризации имеет большой разброс и маленькое смещение. Бэггинг уменьшает разброс алгоритма, поэтому ошибка бэггинга меньше, чем ошибка одного решающего дерева.

Попробуем помимо случайных объектов в подвыборках выбирать и случайные признаки. То есть каждое дерево обучать не на всех признаках, а на случайном подмножестве признаков.

```
In [ ]: res = np.zeros(X_test.shape[0])
        N = 100
        clf = DecisionTreeRegressor()
        for i in range(N):
            ind = np.random.choice(np.arange(X_train.shape[0]),
                                   size = X_train.shape[0], replace = True)
            ind_feat = np.random.choice(np.arange(X_train.shape[1]),
                                        size = round(X_train.shape[1]*0.2))
            clf.fit(X_train[ind][:, ind_feat], y_train[ind])
            res += clf.predict(X_test[:, ind_feat])
```

```
In [ ]: mean_squared_error(y_test, res/N)
```

```
Out[ ]: 3919.174342488953
```

Качество стало заметно хуже.

### 3. Случайный лес

Случайный лес помимо бэггинга над деревьями использует метод случайных подпространств. При построении разбиения в каждой вершине каждого дерева используются не все признаки, а лишь случайное подмножество признаков. Это необходимо, чтобы ответы на построенных деревьях меньше коррелировали между собой.

```
In [ ]: from sklearn.ensemble import RandomForestRegressor
```

```
In [ ]: clf = RandomForestRegressor(n_estimators=100, max_features=0.2)
```

```
In [ ]: clf.fit(X_train, y_train)
```

```
Out[ ]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                               max_features=0.2, max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=100,
                               n_jobs=None, oob_score=False, random_state=None,
                               verbose=0, warm_start=False)
```

```
In [ ]: mean_squared_error(y_test, clf.predict(X_test))
```

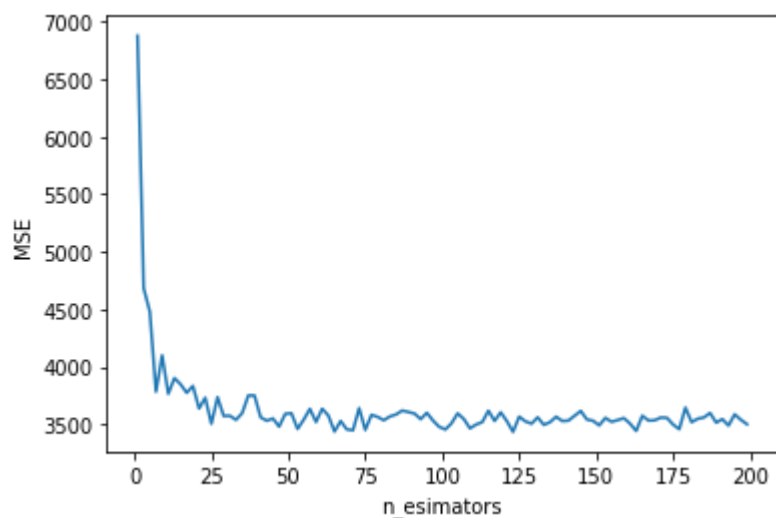
```
Out[ ]: 3623.3489977375566
```

### 4. Зависимость качества от числа деревьев

```
In [ ]: import matplotlib.pyplot as plt
```

```
In [ ]: Q = []
        for n_est in range(1, 200, 2):
            clf = RandomForestRegressor(n_estimators=n_est, max_features=0.2)
            clf.fit(X_train, y_train)
            Q.append(mean_squared_error(y_test, clf.predict(X_test)))
```

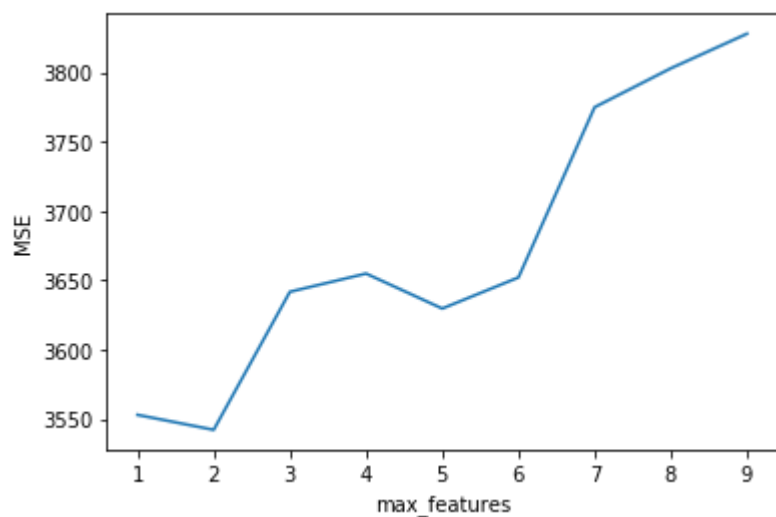
```
In [ ]: plt.plot(range(1, 200, 2), Q)
plt.xlabel('n_estimators')
plt.ylabel('MSE')
plt.show()
```



## 5. Зависимость качества от гиперпараметра max\_features

```
In [ ]: Q = []
for max_feat in range(1, 10):
    clf = RandomForestRegressor(n_estimators=100, max_features=max_feat)
    clf.fit(X_train, y_train)
    Q.append(mean_squared_error(y_test, clf.predict(X_test)))
```

```
In [ ]: plt.plot(range(1, 10), Q)
plt.xlabel('max_features')
plt.ylabel('MSE')
plt.show()
```



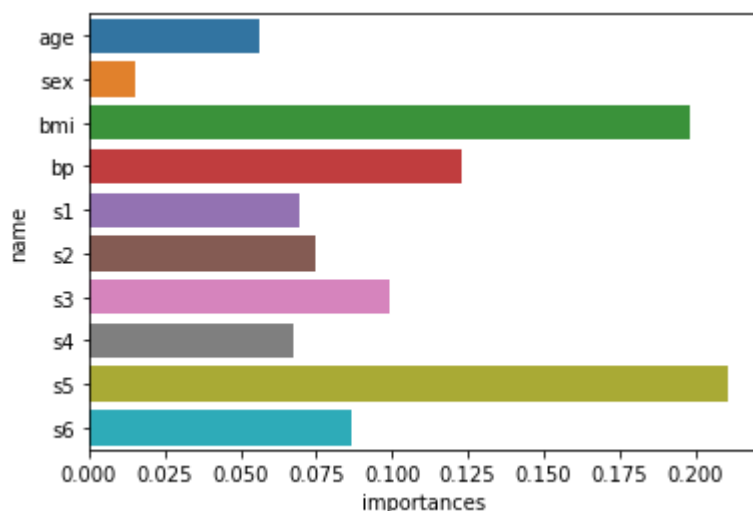
## 6. Важность признаков

```
In [ ]: import seaborn as sns
import pandas as pd
```

```
In [ ]: feature_imp = pd.DataFrame({'importances': clf.feature_importances_, 'name': ds.feature_names})
```

```
In [ ]: sns.barplot(x="importances", y="name", data=feature_imp)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fabc81451d0>
```



Наиболее важные признаки при обучении случайного леса оказались

bmi - Body mass index

s5

Удалим один из наиболее важных признаков

```
In [ ]: ds.feature_names
```

```
Out[ ]: ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

```
In [ ]: ind = np.append(np.arange(2), np.arange(3, 10))
```

```
In [ ]: clf = RandomForestRegressor(n_estimators=100, max_features=0.2)
```

```
In [ ]: clf.fit(X_train[:, ind], y_train)
```

```
Out[ ]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                               max_features=0.2, max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=100,
                               n_jobs=None, oob_score=False, random_state=None,
                               verbose=0, warm_start=False)
```

```
In [ ]: mean_squared_error(y_test, clf.predict(X_test[:, ind]))
```

```
Out[ ]: 3973.0036556561085
```

Удалим наименее важный признак

```
In [ ]: ind = np.append(np.arange(1), np.arange(2, 10))
```

```
In [ ]: clf.fit(X_train[:, ind], y_train)
```

```
Out[ ]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                               max_features=0.2, max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=100,
                               n_jobs=None, oob_score=False, random_state=None,
                               verbose=0, warm_start=False)
```

```
In [ ]: mean_squared_error(y_test, clf.predict(X_test[:, ind]))
```

```
Out[ ]: 3633.3767447963796
```

Видно, что при удалении важных признаков MSE сильно возрастает. При удалении наименее важных признаков MSE почти не изменяется.

## 7. Смешивание моделей

Часто с помощью линейной комбинации разных моделей можно добиться лучшего качества, чем с помощью каждой модели по отдельности.

Используем линейную комбинацию

```
In [ ]: from sklearn.linear_model import LinearRegression
```

```
In [ ]: lr = LinearRegression()
```

```
In [ ]: lr.fit(X_train, y_train)
```

```
Out[ ]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
In [ ]: mean_squared_error(y_test, lr.predict(X_test))
```

```
Out[ ]: 3118.2895401567
```

```
In [ ]: clf = RandomForestRegressor(n_estimators=200)
        clf.fit(X_train, y_train)
```

```
Out[ ]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                               max_features='auto', max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=200,
                               n_jobs=None, oob_score=False, random_state=None,
                               verbose=0, warm_start=False)
```

```
In [ ]: min_mse = mean_squared_error(y_test, lr.predict(X_test))
        for i in range(100):
            mse = mean_squared_error(y_test,
                                     (i*lr.predict(X_test) + (100 - i)*clf.predict(X_test))/100.0)
            if mse < min_mse:
                min_mse = mse
                best_i = i
```



```
In [ ]: mean_squared_error(y_test,  
                           (best_i*lr.predict(X_test) + (100 - best_i)*clf.predict(X_test))/1  
                           00.0)
```

Out[ ]: 3117.577858587075

In [ ]: