Лабораторная работа №4 по курсу "Технологии машинного обучения"

Выполнила Попова Дарья, студентка группы РТ5-61Б

Линейные модели, SVM и деревья решений

Задача регрессии

Продолжим использовать датасет из ЛР №2-3 с данными американских граждан, в котором независимыми переменными являются:

- пол,
- возраст,
- индекс массы тела,
- число детей,
- является ли человек курильщиком (бинарный признак),
- регион проживания (территория США поделена на 4 части),

а целевым признаком - стоимость медицинской страховки (charges).

In [1]:

```
import pandas as pd
import numpy as np
insurance = pd.read_csv('C:\\Users\\Дасупс\\Downloads\\insurance.csv')
```

In [2]:

```
insurance.head()
```

Out[2]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

In [3]:

```
insurance.shape
```

Out[3]:

```
(1338, 7)
```

Предобработка данных

In [4]:

```
# убедимся в том, что в данных нет пропусков
insurance.isnull().any()
```

Out[4]:

```
age         False
sex         False
bmi         False
children    False
smoker      False
region      False
charges     False
dtype: bool
```

In [5]:

```
# закодируем One-Hot Encoding'ом категориальные признаки
insurance = pd.get_dummies(insurance)
```

In [6]:

```
# отмасштабируем числовые признаки со значениями возраста и индексом массы тела
from sklearn.preprocessing import MinMaxScaler
minmax_scaler = MinMaxScaler()
insurance[['age']] = minmax_scaler.fit_transform(insurance[['age']])
insurance[['bmi']] = minmax_scaler.fit_transform(insurance[['bmi']])
```

In [7]:

```
insurance
```

Out[7]:

	age	bmi	children	charges	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	region_southea
0	0.021739	0.321227	0	16884.92400	1	0	0	1	0	0	
1	0.000000	0.479150	1	1725.55230	0	1	1	0	0	0	
2	0.217391	0.458434	3	4449.46200	0	1	1	0	0	0	
3	0.326087	0.181464	0	21984.47061	0	1	1	0	0	1	
4	0.304348	0.347592	0	3866.85520	0	1	1	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	
1333	0.695652	0.403820	3	10600.54830	0	1	1	0	0	1	
1334	0.000000	0.429379	0	2205.98080	1	0	1	0	1	0	
1335	0.000000	0.562012	0	1629.83350	1	0	1	0	0	0	
1336	0.065217	0.264730	0	2007.94500	1	0	1	0	0	0	
1337	0.934783	0.352704	0	29141.36030	1	0	0	1	0	1	

1338 rows × 12 columns

Разделение данных

Для начала разделим данные на независимые переменные и целевой признак.

In [8]:

```
X = insurance.drop(['charges'], axis=1)
y = insurance.charges
```

In [9]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

Построение и обучение моделей

Линейная модель

In [10]:

```
from sklearn.linear_model import LinearRegression
lin_regr = LinearRegression()
lin_regr.fit(X_train, y_train)
lin_regr.coef_, lin_regr.intercept_
```

Out[10]:

```
(array([ 11942.68712197,  12630.55300389,    426.50272274,    -22.81121375,
           22.81121375, -11815.19949515,  11815.19949515,    499.19593062,
          144.04730929,   -282.1724583 ,   -361.07078161]),
 9296.876435444969)
```

Можем обратить внимание на то, какими огромными получились коэффициенты регрессии.

In [11]:

```
y_predicted = lin_regr.predict(X_test)
```

In [12]:

```
from sklearn.metrics import r2_score, mean_squared_error, mean_squared_log_error
# тут он мне почему-то не разрешил использовать логарифм от MSE, поскольку в целевой ф-ии есть отрицательные з
# во-первых, как наличие отрицательных значений может повлиять на квадрат разности
# во-вторых, откуда взяться отрицательным значениям в денежном признаке

# ValueError: Mean Squared Logarithmic Error cannot be used when targets contain negative values.

r2_score(y_test, y_predicted), round(mean_squared_error(y_test, y_predicted),2)
```

Out[12]:

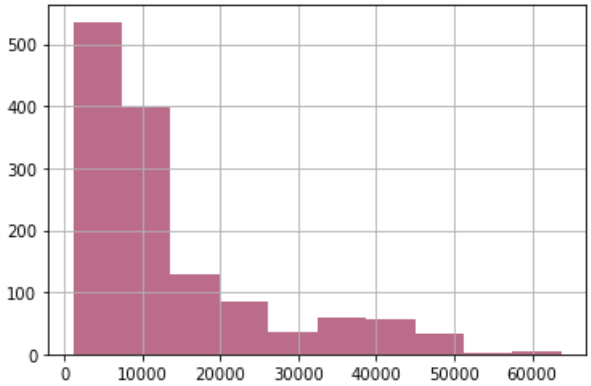
```
(0.7672642952734356, 35117755.74)
```

In [13]:

```
y.hist(color='#bb6c8a')
```

Out[13]:

```
<AxesSubplot:>
```



In [14]:

```
import matplotlib.pyplot as plt
```

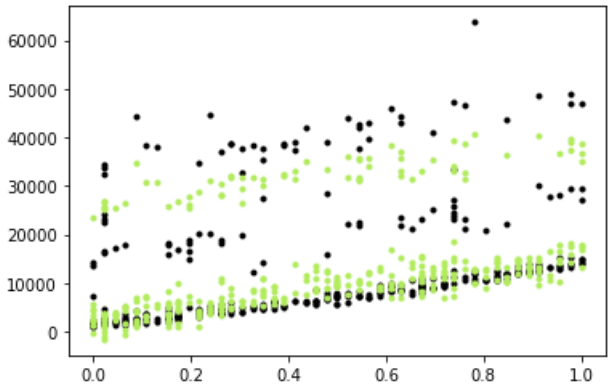
Построим распределение признака "возраст" и на том же графике укажем предсказанные значения.

Здесь и далее чёрными точками обозначаются образцы тестовой выборки (истинные значения), а цветными точками - предсказанные.

In [15]:

```
plt.plot(X_test.age, y_test, 'b.', color='black')
plt.plot(X_test.age, y_predicted, 'b.', color='#b2ec5d')

plt.show()
```

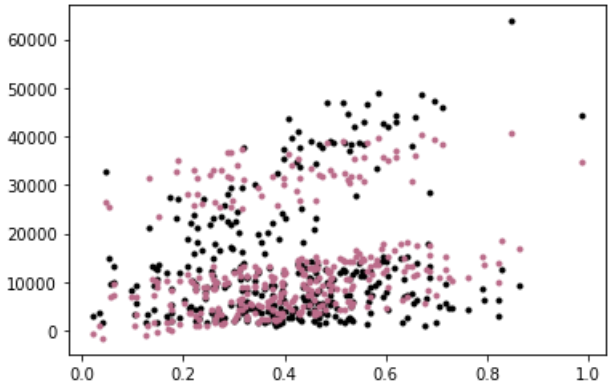


То же сделаем с признаком "ИМТ" (индекс массы тела).

In [16]:

```
plt.plot(X_test.bmi, y_test, 'b.', color='black')
plt.plot(X_test.bmi, y_predicted, 'b.', color='#bb6c8a')

plt.show()
```



И по огромным коэффициентам, и по построенным графикам видно, что линейная модель, скорее всего, очень сильно переобучилась. Воспользуемся регуляризацией Тихонова, чтобы это исправить.

Регуляризация линейной модели

Регуляризация L2 Тихонова

In [17]:

```
from sklearn.linear_model import Ridge
```

In [18]:

```
lin_regr_rigde = Ridge(alpha=1).fit(X_train, y_train)
```

In [19]:

```
lin_regr_rigde.coef_, lin_regr_rigde.intercept_
```

Out[19]:

```
(array([ 11840.74495023,  12152.33475932,    428.75946883,    -24.92954724,
           24.92954724, -11776.5482026 ,  11776.5482026 ,    476.97661247,
          127.91272476,   -248.33656827,   -356.55276896]),
 9505.580901845507)
```

In [20]:

```
ridge_predicted = lin_regr_rigde.predict(X_test)
```

In [21]:

```
r2_score(y_test, y_predicted), round(mean_squared_error(y_test, ridge_predicted),2)
```

Out[21]:

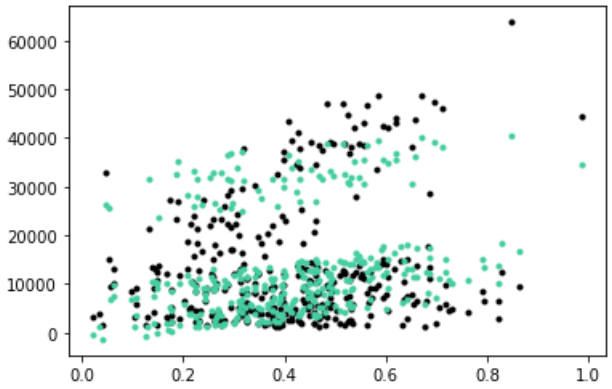
```
(0.7672642952734356, 35128745.49)
```

Стоит отметить, что метрики качества для регуляризации Тихонова при альфа=1 полностью совпадают с метриками для обычной линейной регрессии.

In [22]:

```
plt.plot(X_test.bmi, y_test, 'b.', color='black')
plt.plot(X_test.bmi, ridge_predicted, 'b.', color='#45cea2')

plt.show()
```



Попробуем найти оптимальный коэффициент (гиперпараметр) альфа с помощью решётчатого поиска.

In [23]:

```
from sklearn.model_selection import GridSearchCV
```

In [24]:

```
params = {'alpha':np.geomspace(0.1, 100, 100)}
```

In [25]:

```
grid_search = GridSearchCV(Ridge(), params)
grid_search.fit(X, y)
grid_search.best_params_
```

Out[25]:

```
{'alpha': 0.6135907273413173}
```

In [26]:

```
grid_search.best_estimator_.fit(X,y)
l2_predicted = grid_search.best_estimator_.predict(X_test)
r2_score(y_test, l2_predicted)
```

Out[26]:

```
0.7686233668905967
```

Поиграемся с увеличением весового коэффициента альфа и попробуем сделать слагаемое, накладывающее "штраф" на слишком большие коэффициенты, как можно более значимым в задаче минимизации функции потерь.

In [27]:

```
# альфа = 100
lin_regr_rigde = Ridge(alpha=100).fit(X_train, y_train)
lin_regr_rigde.coef_, lin_regr_rigde.intercept_
```

Out[27]:

```
(array([ 5908.04027169,  2741.06821678,   530.97289044,  -117.92593285,
         117.92593285, -8962.53568254,  8962.53568254,    64.12968384,
        -173.75004231,   345.53513594,  -235.91477746]),
 14153.734004186845)
```

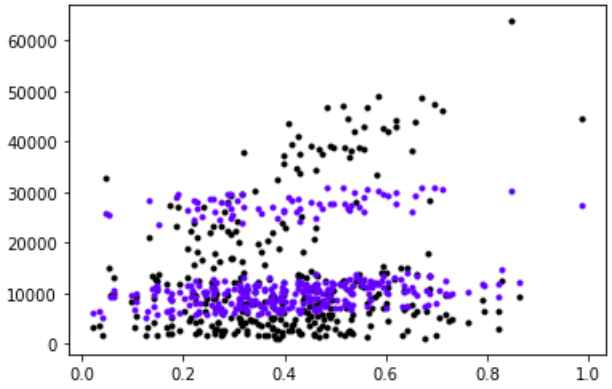
In [28]:

```
ridge_predicted = lin_regr_rigde.predict(X_test)
```

In [29]:

```
plt.plot(X_test.bmi, y_test, 'b.', color='black')
plt.plot(X_test.bmi, ridge_predicted, 'b.', color='#6600ff')

plt.show()
```



In [30]:

```
r2_score(y_test, ridge_predicted), round(mean_squared_error(y_test, ridge_predicted),2)
```

Out[30]:

```
(0.6841665894438855, 47656463.28)
```

In [31]:

```
# альфа = 1000
lin_regr_rigde = Ridge(alpha=1000).fit(X_train, y_train)
lin_regr_rigde.coef_, lin_regr_rigde.intercept_
```

Out[31]:

```
(array([ 1005.26705866,   364.51650021,   436.65706518,  -131.32871284,
         131.32871284, -2870.63084621,  2870.63084621,    14.50062312,
        -107.17980267,   192.10049214,   -99.42131259]),
 13866.184326479288)
```

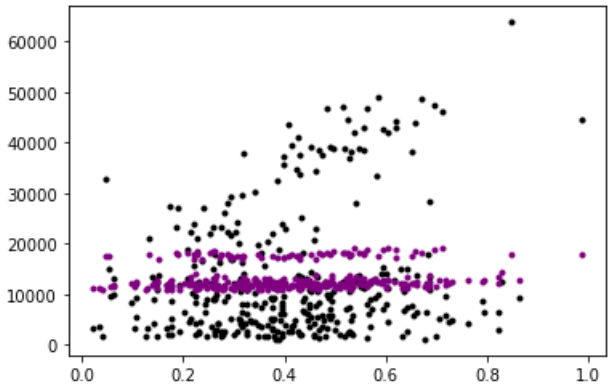
In [32]:

```
ridge_predicted = lin_regr_rigde.predict(X_test)
```

In [33]:

```
plt.plot(X_test.bmi, y_test, 'b.', color='black')
plt.plot(X_test.bmi, ridge_predicted, 'b.', color='purple')

plt.show()
```



In [34]:

```
r2_score(y_test, ridge_predicted), round(mean_squared_error(y_test, ridge_predicted),2)
```

Out[34]:

```
(0.289070406758723, 107272976.57)
```

In [35]:

```
# альфа = 10000
lin_regr_rigde = Ridge(alpha=10000).fit(X_train, y_train)
lin_regr_rigde.coef_, lin_regr_rigde.intercept_
```

Out[35]:

```
(array([ 107.28776271,   38.05730053,  100.39646239,  -24.0991946 ,
          24.0991946 , -369.40327403,  369.40327403,    2.62380861,
         -15.02318135,   26.9101421 ,  -14.51076936]),
 13309.656315564356)
```

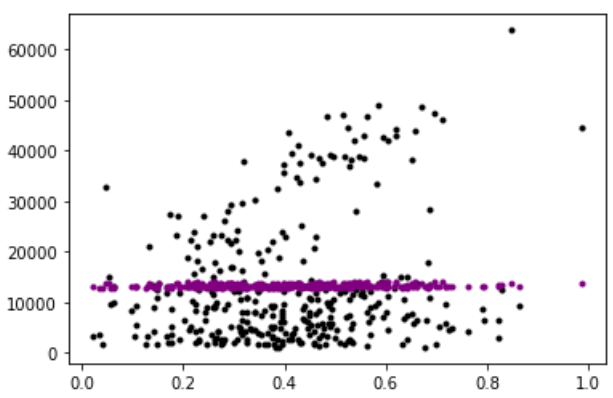
In [36]:

```
ridge_predicted = lin_regr_rigde.predict(X_test)
```

In [37]:

```
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, ridge_predicted, 'b.', color='purple')

plt.show()
```



Можно заметить, как при постоянном увеличении гиперпараметра альфа множество предсказанных точек всё больше и больше сжимается в прямую линию.

In [38]:

```
r2_score(y_test, ridge_predicted), round(mean_squared_error(y_test, ridge_predicted),2)
```

Out[38]:

```
(0.0415245196396451, 144625457.61)
```

Итак, можно сделать следующие выводы:

- При гиперпараметре альфа = 10000, когда наша линейная модель наиболее похожа на график прямой, модель показывает худшие метрики.
- Достаточно высокий коэффициент детерминации модель показывает или на обычной линейной регрессии, или с регуляризацией Тихонова с гиперпараметром альфа = 1.
- Оптимальным гиперпараметром является альфа = 0.6.
- В целом уже можно сказать, что линейная модель слабо описывает наши данные.

Регуляризация L1 (LASSO)

In [39]:

```
from sklearn.linear_model import Lasso
```

In [40]:

```
lasso = Lasso(alpha=0.1).fit(X_train, y_train)
lasso_predicted = lasso.predict(X_test)
r2_score(y_test, lasso_predicted)
```

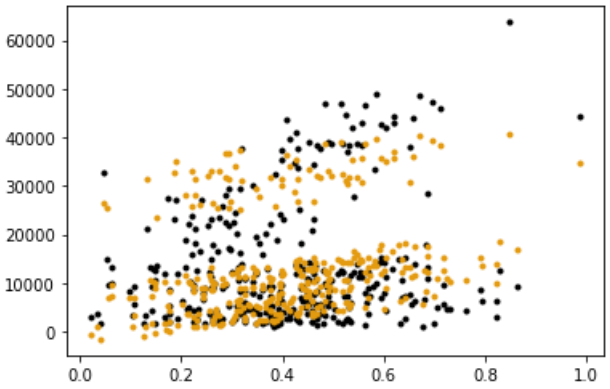
Out[40]:

```
0.7672630204998246
```

In [41]:

```
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, lasso_predicted, 'b.', color='#e49b0f')

plt.show()
```



In [42]:

```
lasso = Lasso(alpha=100).fit(X_train, y_train)
lasso_predicted = lasso.predict(X_test)
r2_score(y_test, lasso_predicted)
```

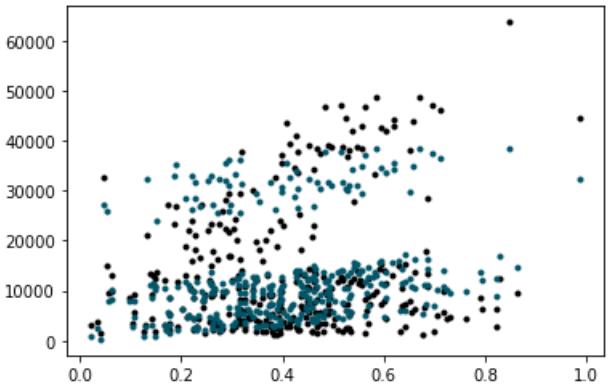
Out[42]:

```
0.7620847427511432
```

In [43]:

```
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, lasso_predicted, 'b.', color='#025669')

plt.show()
```



In [44]:

```
lasso = Lasso(alpha=1000).fit(X_train, y_train)
lasso_predicted = lasso.predict(X_test)
r2_score(y_test, lasso_predicted)
```

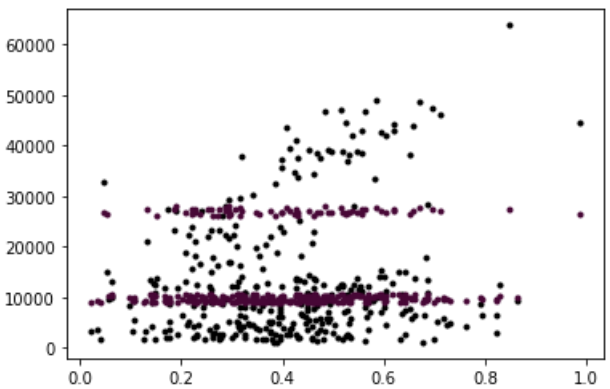
Out[44]:

```
0.6121964185679767
```

In [45]:

```
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, lasso_predicted, 'b.', color='#470736')

plt.show()
```



Найдём оптимальный параметр с решётчатым поиском.

In [46]:

```
params = {'alpha' : np.arange(0.01, 10, 0.5)}

lasso_grid_search = GridSearchCV(Lasso(max_iter=10000), params, scoring='neg_mean_squared_error')
```

```
lasso_grid_search.fit(X, y)
lasso_grid_search.best_params_
```

Out[46]:

```
{'alpha': 8.01}
```

In [47]:

```
lasso_grid_search.best_estimator_.fit(X_train, y_train)
lasso_grid_predicted = lasso_grid_search.best_estimator_.predict(X_test)
r2_score(y_test, lasso_grid_predicted)
```

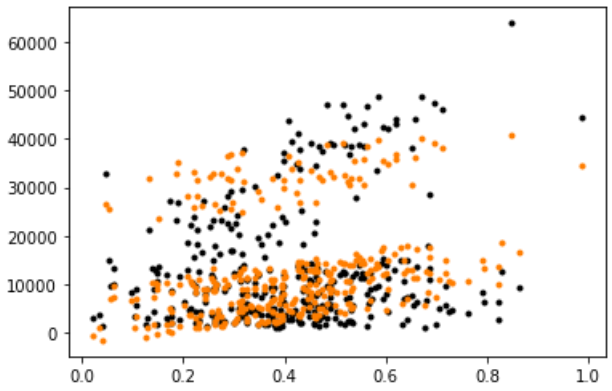
Out[47]:

```
0.7671235608763378
```

In [48]:

```
lasso = Lasso(alpha=8).fit(X_train, y_train)
lasso_predicted = lasso.predict(X_test)
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, lasso_predicted, 'b.', color='#ff7e00')

plt.show()
```



ElasticNet

И теперь поиграемся с ElasticNet, позволяющей проводить одновременную l1 и l2 регуляризацию.

In [49]:

```
from sklearn.linear_model import ElasticNet
```

Сначала возьмём произвольные параметры.

In [50]:

```
elastic = ElasticNet(alpha=0.1, l1_ratio = 0.5)
```

In [51]:

```
elastic.fit(X_train, y_train)
elnet_predicted = elastic.predict(X_test)
r2_score(y_test, elnet_predicted)
```

Out[51]:

```
0.7301417699541504
```

Теперь найдём оптимальные значения гиперпараметров альфа и l1_ratio.

In [52]:

```
params = [{'alpha':np.arange(0.1,10,0.5), 'l1_ratio':np.arange(0.1,0.9,0.1)}]
```

In [53]:

```
elnet_grid = GridSearchCV(ElasticNet(max_iter=10000), params)
elnet_grid.fit(X_train, y_train)
elnet_grid.best_params_
```

Out[53]:

```
{'alpha': 0.1, 'l1_ratio': 0.8}
```

In [54]:

```
elnet_grid.best_estimator_.fit(X_train, y_train)
best_elnet_predicted = elnet_grid.best_estimator_.predict(X_test)
r2_score(y_test, best_elnet_predicted)
```

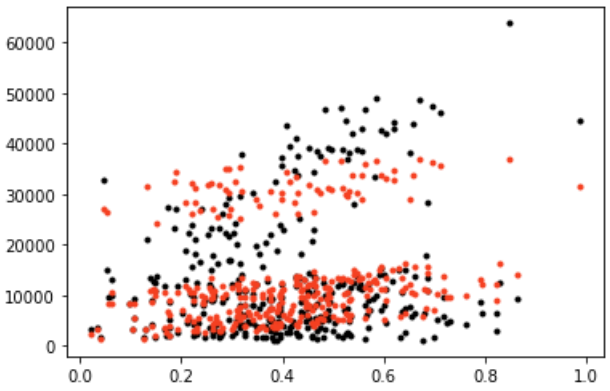
Out[54]:

```
0.7563164348609223
```

In [55]:

```
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, best_elnet_predicted, 'b.', color='#f54021')
```

```
plt.show()
```



Дерево решений

In [56]:

```
from sklearn.tree import DecisionTreeRegressor
```

Сначала обучим дерево с какой-нибудь достаточно большой глубиной.

In [57]:

```
tree_regressor_long = DecisionTreeRegressor(criterion='mse', max_depth=15)
tree_regressor_long.fit(X_train, y_train)
y_long_predicted = tree_regressor_long.predict(X_test)

r2_score(y_test, y_long_predicted)
```

Out[57]:

```
0.6923841599227982
```

Теперь, наоборот, "подстрижём" дерево так, чтобы максимальная глубина равнялась двум.

In [58]:

```
tree_regressor = DecisionTreeRegressor(criterion='mse', max_depth=2)
```

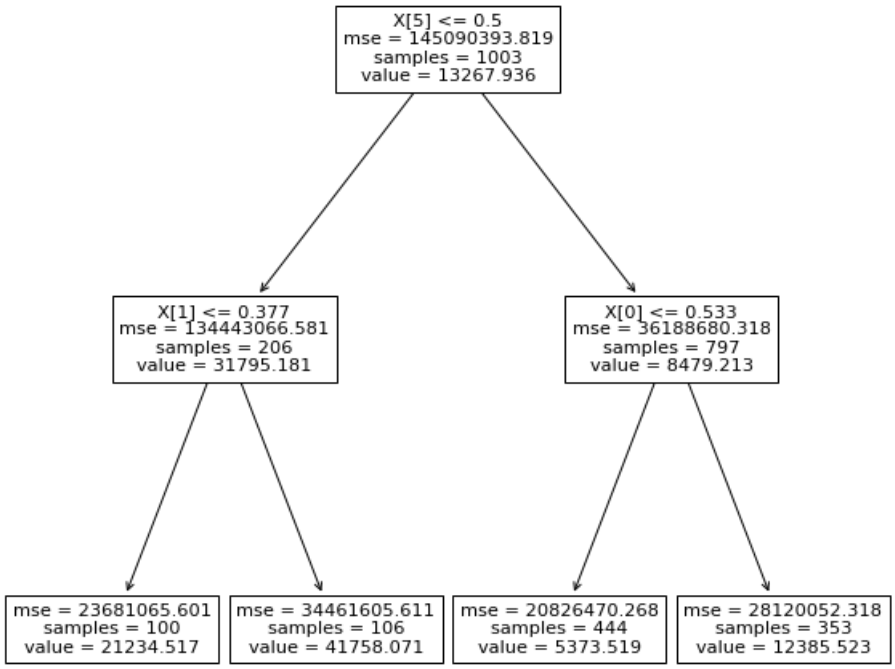
In [59]:

```
tree_regressor.fit(X_train, y_train)
tree_predicted = tree_regressor.predict(X_test)
```

Визуализируем полученное дерево.

In [60]:

```
from sklearn import tree
plt.figure(figsize=(10,10))
tree.plot_tree(tree_regressor)
plt.show()
```



In [61]:

```
# посмотрим, какой признак стоит в корне дерева
X.columns.values[5]
```

Out[61]:

```
'smoker_no'
```

In [62]:

```
# посмотри на метрики качества модели
r2_score(y_test, tree_predicted), mean_squared_log_error(y_test, tree_predicted), round(mean_squared_error(y_t
```

Out[62]:

```
(0.8210441258582764, 0.2828134377526337, 27002855.81)
```

С помощью решётчатого поиска попробуем определить оптимальную глубину нашего дерева.

In [63]:

```
# params = {'max_depth':range(1, 20), 'min_samples_split':range(2,20), 'min_samples_leaf':range(1,10)}
params = {'max_depth':range(1, 20)}
tree_grid_search = GridSearchCV(DecisionTreeRegressor(), params, cv=5, scoring='r2')
tree_grid_search.fit(X, y)
tree_grid_search.best_params_
```

Out[63]:

```
{'max_depth': 4}
```

In [64]:

```
params = [{'min_samples_leaf':range(1, 20)}]
tree_grid_search = GridSearchCV(DecisionTreeRegressor(), params, cv=5, scoring='r2')
tree_grid_search.fit(X, y)
tree_grid_search.best_params_
```

Out[64]:

```
{'min_samples_leaf': 18}
```

In [65]:

```
params = [{'min_samples_split':range(2, 20)}]
tree_grid_search = GridSearchCV(DecisionTreeRegressor(), params, cv=5, scoring='r2')
tree_grid_search.fit(X, y)
tree_grid_search.best_params_
```

Out[65]:

```
{'min_samples_split': 19}
```

In [66]:

```
best_tree_regr = DecisionTreeRegressor(max_depth=4, min_samples_leaf=18, random_state=42)
best_tree_regr.fit(X_train, y_train)
best_tree_predicted = best_tree_regr.predict(X_test)
```

```
r2_score(y_test, best_tree_predicted), mean_squared_log_error(y_test, best_tree_predicted)
```

Out[66]:

```
(0.8467998179620155, 0.20026717144779713)
```

Метрики для оптимальной модели достаточно хорошо говорят о её качестве.

In [67]:

```
# важность признаков
list(zip(X.columns.values, best_tree_regr.feature_importances_))
```

Out[67]:

```
[('age', 0.11738407352511335),
 ('bmi', 0.1749369455566289),
 ('children', 0.0029484138015523644),
 ('sex_female', 0.0),
 ('sex_male', 0.0),
 ('smoker_no', 0.7047305671167053),
 ('smoker_yes', 0.0),
 ('region_northeast', 0.0),
 ('region_northwest', 0.0),
 ('region_southeast', 0.0),
 ('region_southwest', 0.0)]
```

In [68]:

```
# убедимся, что в сумме дают единицу
sum(best_tree_regr.feature_importances_)
```

Out[68]:

```
1.0
```

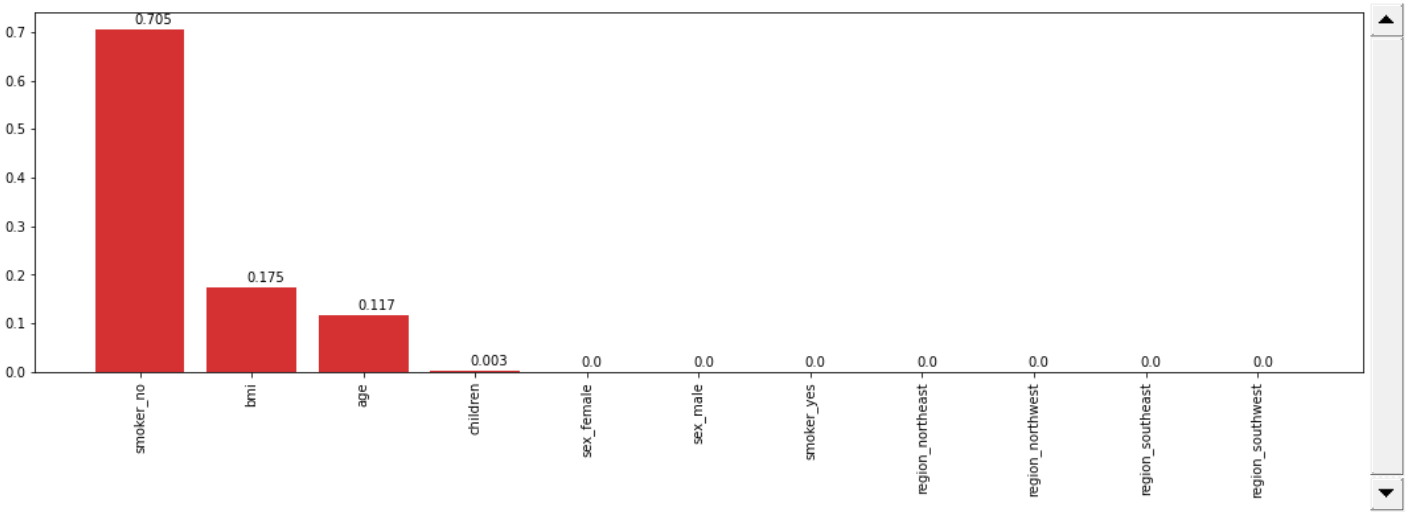
In [69]:

```
# воспользуемся блоком кода, приведённым Юрием Евгеньевичем в лекции по деревьям
# https://nbviewer.jupyter.org/github/ugapanyuk/ml_course_2021/blob/main/common/notebooks/trees/trees.ipynb


from operator import itemgetter

def draw_feature_importances(tree_model, X_dataset, figsize=(18,5)):
    # 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, color='#d53032')
    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

insurance_tree_cl_fl, insurance_tree_cl_fd = draw_feature_importances(best_tree_regr, X)
```



SVM

LinearSVR

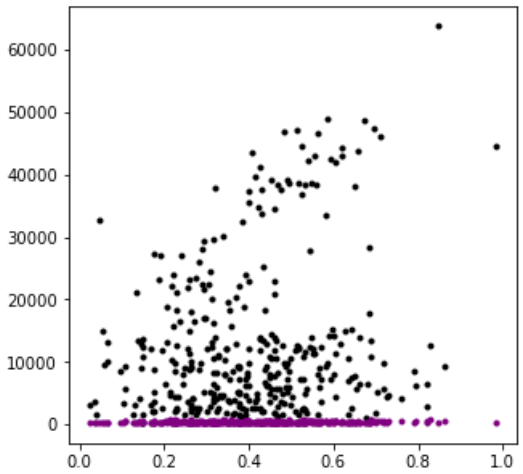
In [70]:

```
from sklearn.svm import LinearSVR
```

In [71]:

```
# обучим модель с C=0.1
lin_svr = LinearSVR(C=0.1,  max_iter=10000)
lin_svr.fit(X_train, y_train)
lin_svr_predicted = lin_svr.predict(X_test)

fig, ax = plt.subplots(figsize=(5,5))
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, lin_svr_predicted, 'b.', color='purple', linewidth=3)
plt.show()
```



In [72]:

```
r2_score(y_test, lin_svr_predicted)
```

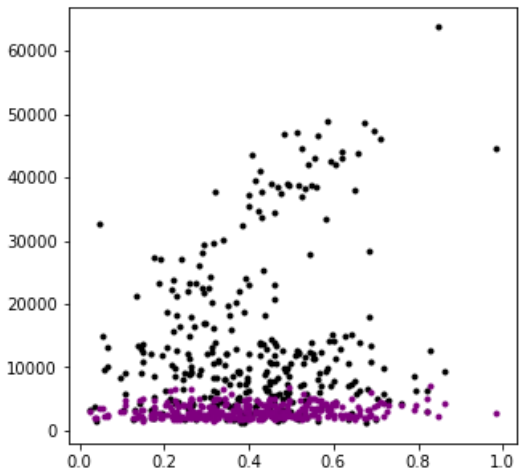
Out[72]:

```
-1.1008377236212334
```

In [73]:

```
# обучим модель с C=1
lin_svr = LinearSVR(C=1.0,  max_iter=10000)
lin_svr.fit(X_train, y_train)
lin_svr_predicted = lin_svr.predict(X_test)

fig, ax = plt.subplots(figsize=(5,5))
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, lin_svr_predicted, 'b.', color='purple', linewidth=3)
plt.show()
```



In [74]:

```
lin_svr.coef_, lin_svr.intercept_
```

Out[74]:

```
(array([451.49048672, 305.4108431 , 889.3568035 , 420.25401058,
        349.36975647, 563.62376705, 206.        , 218.25401058,
        200.        , 179.        , 172.36975647]),
 array([769.62376705]))
```

In [75]:

```
r2_score(y_test, lin_svr_predicted)
```

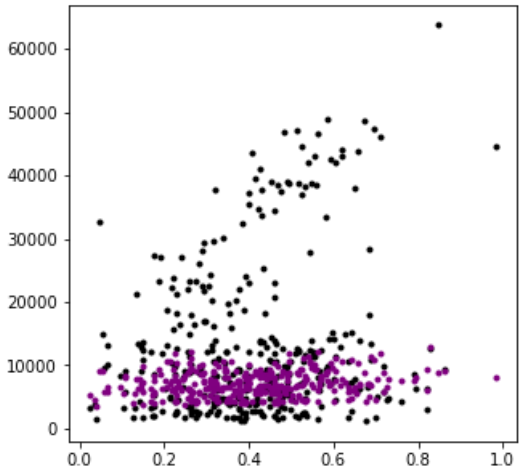
Out[75]:

```
-0.6972773114706317
```

In [76]:

```
# обучим модель с C=10
lin_svr = LinearSVR(C=10.0,  max_iter=10000)
lin_svr.fit(X_train, y_train)
lin_svr_predicted = lin_svr.predict(X_test)

fig, ax = plt.subplots(figsize=(5,5))
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, lin_svr_predicted, 'b.', color='purple', linewidth=3)
plt.show()
```



In [77]:

```
r2_score(y_test, lin_svr_predicted)
```

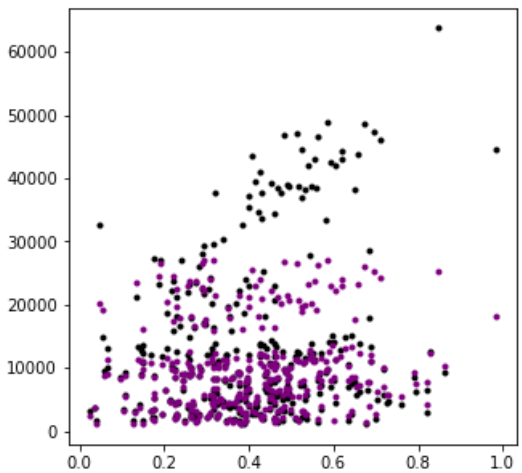
Out[77]:

```
-0.11244330217921705
```

In [78]:

```
# обучим модель с C=100
lin_svr = LinearSVR(C=100.0, max_iter=10000)
lin_svr.fit(X_train, y_train)
lin_svr_predicted = lin_svr.predict(X_test)

fig, ax = plt.subplots(figsize=(5,5))
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, lin_svr_predicted, 'b.', color='purple')
plt.show()
```



In [79]:

```
lin_svr.coef_, lin_svr.intercept_
```

Out[79]:

```
(array([10779.06206807,  1621.46872852,   479.18680278,  1977.07299238,
         1676.33845486, -5546.58855276,  9200.        ,  1280.38381248,
         1014.80505975,   679.96722911,   678.25534591]),
 array([3653.41144724]))
```

In [80]:

```
r2_score(y_test, lin_svr_predicted)
```

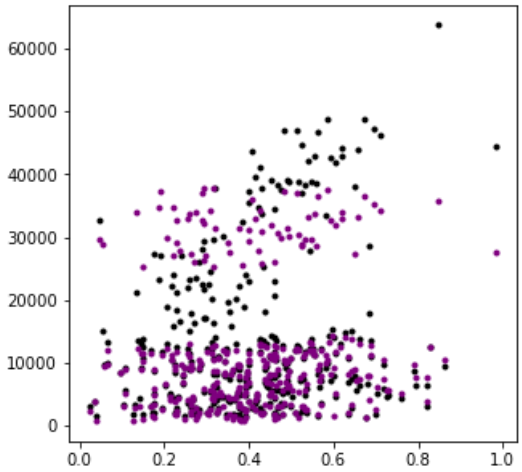
Out[80]:

```
0.5699236321766927
```

In [81]:

```
# обучим модель с C=1000
lin_svr = LinearSVR(C=1000.0, max_iter=10000)
lin_svr.fit(X_train, y_train)
lin_svr_predicted = lin_svr.predict(X_test)

fig, ax = plt.subplots(figsize=(5,5))
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, lin_svr_predicted, 'b.', color='purple')
plt.show()
```



In [82]:

```
lin_svr.coef_, lin_svr.intercept_
```

Out[82]:

```
(array([11923.34812182,  1715.33434635,   384.8154901 ,  3037.69524802,
         2606.18692892, -9432.66618979, 15076.54836674,  1816.4632129 ,
         1491.51025198,  1217.46628088,  1118.44243118]),
 array([5643.88217695]))
```

In [83]:

```
r2_score(y_test, lin_svr_predicted)
```

Out[83]:

```
0.7372168970886481
```

Теперь найдём оптимальный гиперпараметр для линейной SVR.

In [84]:

```
params = [{'C':np.geomspace(0.01,1000, 20)}]
```

In [86]:

```
lin_svr_grid = GridSearchCV(LinearSVR(max_iter=10000), params)
lin_svr_grid.fit(X, y)
lin_svr_grid.best_params_
```

Out[86]:

```
{'C': 1000.0}
```

SVR (kernel trick)

In [87]:

```
from sklearn.svm import SVR
```

Радиально-базисная функция.

In [88]:

```
rbf_params = [{'gamma': np.arange(0.1, 0.9, 0.1)}]
```

In [89]:

```
grid_search_rbf = GridSearchCV(SVR(kernel='rbf'), rbf_params, scoring='neg_mean_squared_error')
grid_search_rbf.fit(X, y)
grid_search_rbf.best_params_
```

Out[89]:

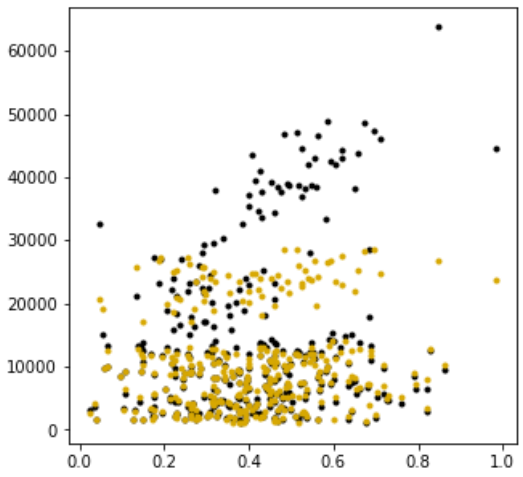
```
{'gamma': 0.2}
```

In [90]:

```
rbf_svr = SVR(kernel='rbf', gamma=0.2, C=1000.0)
rbf_svr.fit(X_train, y_train)
rbf_predicted = rbf_svr.predict(X_test)
```

In [91]:

```
fig, ax = plt.subplots(figsize=(5,5))
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, rbf_predicted, 'b.', color='#d8a903')
plt.show()
```



In [92]:

```
r2_score(y_test, rbf_predicted)
```

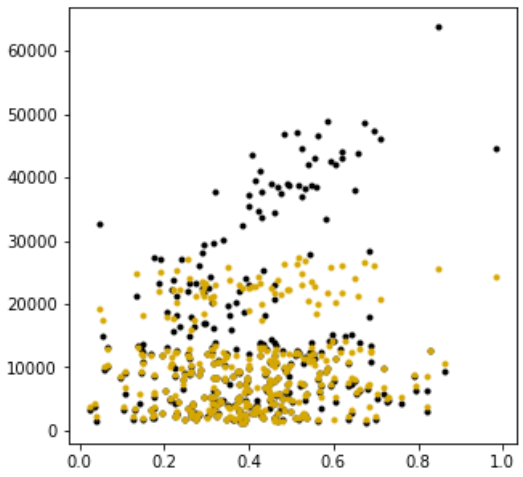
Out[92]:

```
0.6237047686584283
```

In [98]:

```
rbf_svr = SVR(kernel='rbf', gamma=0.7, C=1000.0)
rbf_svr.fit(X_train, y_train)
rbf_predicted = rbf_svr.predict(X_test)

fig, ax = plt.subplots(figsize=(5,5))
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, rbf_predicted, 'b.', color='#d8a903')
plt.show()
```



In [99]:

```
r2_score(y_test, rbf_predicted)
```

Out[99]:

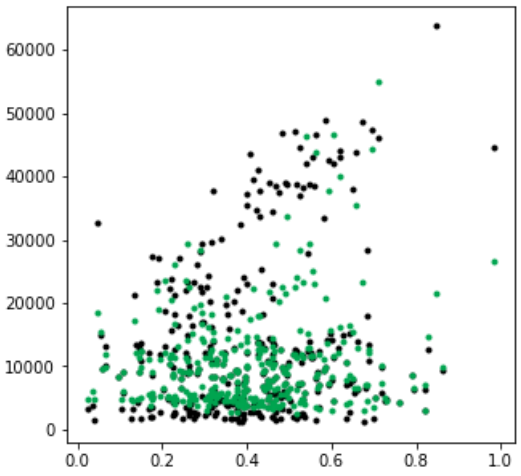
```
0.5859766766580733
```

Полиномиальная функция.

In [100]:

```
poly_svr = SVR(kernel='poly', degree=4, gamma=0.2, C=1000.0)
poly_svr.fit(X_train, y_train)
poly_predicted = poly_svr.predict(X_test)

fig, ax = plt.subplots(figsize=(5,5))
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, poly_predicted, 'b.', color='#00a550')
plt.show()
```



In [101]:

```
r2_score(y_test, poly_predicted)
```

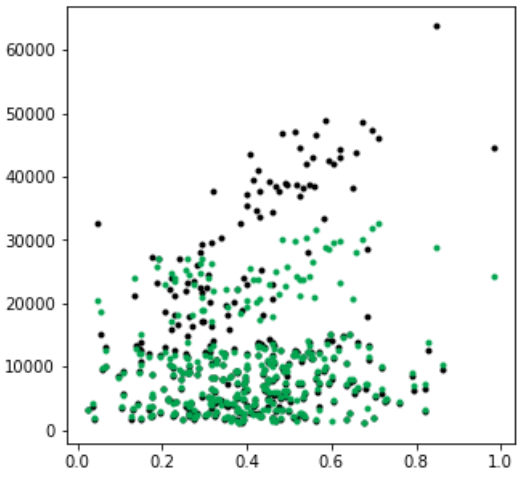
Out[101]:

```
0.5820604452047222
```

In [106]:

```
poly_svr = SVR(kernel='poly', degree=2, gamma=0.2, C=1000.0)
poly_svr.fit(X_train, y_train)
poly_predicted = poly_svr.predict(X_test)

fig, ax = plt.subplots(figsize=(5,5))
plt.plot(X_test.bmi, y_test,'b.', color='black')
plt.plot(X_test.bmi, poly_predicted, 'b.', color='#00a550')
plt.show()
```



In [107]:

```
r2_score(y_test, poly_predicted)
```

Out[107]:

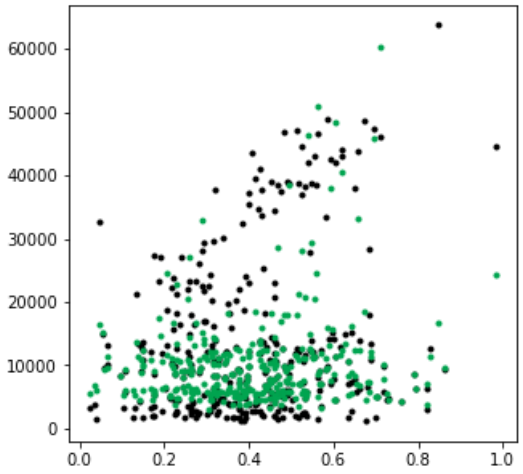
```
0.6537736783349605
```

In [109]:

```
poly_svr = SVR(kernel='poly', degree=5, gamma=0.2, C=1000.0)
poly_svr.fit(X_train, y_train)
poly_predicted = poly_svr.predict(X_test)

fig, ax = plt.subplots(figsize=(5,5))
```

```
plt.plot(X_test.bmi, y_test, 'b.',color='black')
plt.plot(X_test.bmi, poly_predicted, 'b.', color='#00a550')
plt.show()
```



In [110]:

```
r2_score(y_test, poly_predicted)
```

Out[110]:

```
0.4674804946480089
```

Выводы

На выбранном мной наборе данных наилучший результат (метрику R^2) показало дерево решений.

In [ ]:

In [ ]: