Лабораторная работа №3 по курсу "Технологии машинного обучения"

Выполнила Попова Дарья, студентка группы РТ5-61Б

Метод k ближайших соседей, метрики оценки качества модели, кросс-валидация и подбор гиперпараметров для задачи регрессии

In [1]:

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

Будем работать с набором данных американских граждан, в котором фичами являются:

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

На основании этих признаков нужно предсказать целевой - стоимость медицинской страховки (charges).

In [2]:

```
insurance
```

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
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

Предобработка данных

In [3]:

```
insurance.isnull().any()
```

Out[3]:

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

Пропущенных значений в нет ни в одной из колонок, зато мы можем закодировать категориальные признаки.

In [4]:

```
insurance.region.unique()
```

Out[4]:

```
array(['southwest', 'southeast', 'northwest', 'northeast'], dtype=object)
```

In [5]:

```
insurance.sex.unique()
```

Out[5]:

```
array(['female', 'male'], dtype=object)
```

In [6]:

```
insurance.smoker.unique()
```

Out[6]:

```
array(['yes', 'no'], dtype=object)
```

In [7]:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

In [8]:

```
lbl_enc = LabelEncoder()
```

In [9]:

```
# кодируем категориальные колонки для датасета, в котором мы будем масштабировать признаки
insurance = pd.get_dummies(insurance)
```

In [10]:

```
# кодируем категориальные колонки для датасета, в котором мы НЕ будем масштабировать признаки
insurance_unscaled = pd.get_dummies(insurance_unscaled)
```

In [11]:

```
from sklearn.preprocessing import MinMaxScaler
```

In [12]:

```
minmax_scaler = MinMaxScaler()
```

In [13]:

```
insurance[['age']] = minmax_scaler.fit_transform(insurance[['age']])
insurance[['bmi']] = minmax_scaler.fit_transform(insurance[['bmi']])
```

In [14]:

```
insurance
```

Out[14]:

	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

In [15]:

```
insurance_unscaled
```

Out[15]:

	age	bmi	children	charges	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	region_southeast	re
0	19	27.900	0	16884.92400	1	0	0	1	0	0	0	
1	18	33.770	1	1725.55230	0	1	1	0	0	0	1	
2	28	33.000	3	4449.46200	0	1	1	0	0	0	1	
3	33	22.705	0	21984.47061	0	1	1	0	0	1	0	
4	32	28.880	0	3866.85520	0	1	1	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	
1333	50	30.970	3	10600.54830	0	1	1	0	0	1	0	
1334	18	31.920	0	2205.98080	1	0	1	0	1	0	0	
1335	18	36.850	0	1629.83350	1	0	1	0	0	0	1	
1336	21	25.800	0	2007.94500	1	0	1	0	0	0	0	
1337	61	29.070	0	29141.36030	1	0	0	1	0	1	0	

1338 rows × 12 columns

Итак, датасет в порядке! Можем разбивать его на обучающую и тестовую выборки.

Разделение датасета на обучающую и тестовую выборки

In [16]:

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

In [17]:

```
X_unscaled = insurance_unscaled.drop('charges', axis=1)
y_unscaled = insurance_unscaled.charges
```

In [18]:

```
from sklearn.model_selection import train_test_split

# разделяем отмасштабированные данные
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.3, random_state = 42)

# разделяем неотмасштабированные данные
X_train_unscaled, X_test_unscaled, y_train_unscaled, y_test_unscaled = train_test_split(
    X_unscaled, y_unscaled, test_size = 0.3, random_state = 42)
```

Модель для алгоритма k ближайших соседей (для произвольного заданного k)

In [19]:

```
from sklearn.neighbors import KNeighborsRegressor
kneighbors_regressor = KNeighborsRegressor(n_neighbors = 18)

# для отмасштабированных данных
kneighbors_regressor.fit(X_train, y_train)
y_predicted = kneighbors_regressor.predict(X_test)

# для неотмасштабированных данных
kneighbors_regressor.fit(X_train_unscaled, y_train_unscaled)
y_predicted_unscaled = kneighbors_regressor.predict(X_test_unscaled)
```

Оценка качества модели (для произвольно выбранного k)

In [20]:

```
from sklearn.metrics import max_error, mean_absolute_error, mean_squared_error, median_absolute_error,mean_squ
```

In [21]:

```
# рассчитаем метрики качества для отмасштабированных данных
max_err, mean_abs_err, mean_sq_err, median, log_mse = max_error(y_test, y_predicted), \
mean_absolute_error(y_test, y_predicted), mean_squared_error(y_test, y_predicted), \
median_absolute_error(y_test, y_predicted), mean_squared_log_error(y_test, y_predicted)

# и для неотмасштабированных данных
max_err_uns, mean_abs_err_uns, mean_sq_err_uns, median_uns, log_mse_uns = max_error(y_test_unscaled, y_predict
```

```
mean_absolute_error(y_test_unscaled, y_predicted_unscaled), mean_squared_error(y_test_unscaled, y_predicted_u
median_absolute_error(y_test_unscaled, y_predicted_unscaled), mean_squared_log_error(y_test_unscaled, y_predi
```

Сравним метрики качества модели, обученной на отмасштабированный и на неотмасштабированных данных.

In [22]:

```
print("{:<25} {:<15} {:<15} {:<15}".format('metrics','scaled','unscaled', 'difference, %'))
print()
metrics_dict = {'max error':[round(max_err,2), round(max_err_uns,2), str(round(100 * (max_err_uns - max_err) ,
                'mean absolute error':[round(mean_abs_err,2), round(mean_abs_err_uns,2), str(round(100 * (mean_
                'mean squared error':[round(mean_sq_err,2), round(mean_sq_err_uns,2), str(round(100 * (mean_sq_
                'median absolute error': [round(median,2), round(median_uns,2), str(round(100 * (median_uns - m
                'log (MSE)': [round(log_mse,2), round(log_mse_uns,2), str(round(100 * (log_mse_uns - log_mse) ,
for key, value in metrics_dict.items():
    scaled, unscaled, difference = value
    print ("{:<25} {:<15} {:<15} {:<15}".format(key, scaled, unscaled, difference))
```

```
metrics                   scaled          unscaled        difference, %

max error                 32116.26        46226.09        43.93 %
mean absolute error       4396.76         8146.35         85.28 %
mean squared error        43981433.13     117210816.61    166.5 %
median absolute error     2487.77         5728.1          130.25 %
log (MSE)                 0.28            0.67            135.48 %
```

Подбор гиперпараметра k и кросс-валидация

In [23]:

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

In [24]:

```
n_range = np.array(range(1,100,2))
parameters = [{'n_neighbors':n_range}]
grid_search = GridSearchCV(KNeighborsRegressor(), parameters, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
grid_search.best_params_, -grid_search.best_score_
```

Out[24]:

```
({'n_neighbors': 5}, 47875435.724750735)
```

In [25]:

```
n_range = np.array(range(1,10,1))
parameters = [{'n_neighbors':n_range}]
grid_search = GridSearchCV(KNeighborsRegressor(), parameters, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
grid_search.best_params_, -grid_search.best_score_
```

Out[25]:

```
({'n_neighbors': 5}, 47875435.724750735)
```

Получили оптимальное значение гиперпараметра k = 4! Проверим его на 10 фолдах.

In [26]:

```
# используем всё ту же среднюю квадратичную ошибку (MSE), но разобьём на 10 фолдов
grid_search = GridSearchCV(KNeighborsRegressor(), parameters, cv=10, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
grid_search.best_params_, -grid_search.best_score_
```

Out[26]:

```
({'n_neighbors': 5}, 45735758.60557963)
```

In [27]:

```
# используем среднюю абсолютную ошибку (MAE), но разобьём на 10 фолдов
grid_search = GridSearchCV(KNeighborsRegressor(), parameters, cv=10, scoring='neg_mean_absolute_error')
grid_search.fit(X_train, y_train)
grid_search.best_params_, -grid_search.best_score_
```

Out[27]:

```
({'n_neighbors': 2}, 4054.1614569049984)
```

Значение метрики изменилось, а вот наш оптимальный параметр - нет, так что всё отлично!

Попробуем проверить наш результат на другой стратегии кросс-валидации.

In [28]:

```
from sklearn.model_selection import ShuffleSplit, LeavePOut
```

In [29]:

```
# ShuffleSplit
grid_search = GridSearchCV(KNeighborsRegressor(), parameters, cv=ShuffleSplit(n_splits=5, test_size=0.25, ran
grid_search.fit(X_train, y_train)
grid_search.best_params_, -grid_search.best_score_
```

Out[29]:

```
({'n_neighbors': 6}, 50678961.08212725)
```

И используя другую "решающую" метрику качества.

In [30]:

```
# опираемся на среднюю абсолютную ошибку
grid_search = GridSearchCV(KNeighborsRegressor(), parameters, cv=5, scoring='neg_mean_absolute_error')
grid_search.fit(X_train, y_train)
grid_search.best_params_, -grid_search.best_score_
```

Out[30]:

```
({'n_neighbors': 2}, 4255.5316465397655)
```

In [31]:

```
# опираемся на медианную абсолютную ошибку
grid_search = GridSearchCV(KNeighborsRegressor(), parameters, cv=5, scoring='neg_median_absolute_error')
grid_search.fit(X_train, y_train)
grid_search.best_params_, -grid_search.best_score_
```

Out[31]:

```
({'n_neighbors': 1}, 953.8170949999997)
```

In [32]:

```
# опираемся на логарифм средней квадратичной ошибки
grid_search = GridSearchCV(KNeighborsRegressor(), parameters, cv=5, scoring='neg_mean_squared_log_error')
grid_search.fit(X_train, y_train)
grid_search.best_params_, -grid_search.best_score_
```

Out[32]:

```
({'n_neighbors': 7}, 0.25002673792257546)
```

Сравним метрики на модели с оптимальным и случайным гиперпараметром.

In [33]:

```
print("{:<25} {:<15} {:<15} {:<15}".format('метрика','случайный','оптимальный', 'разница, %'))
print()
metrics_dict = {'mean absolute error': [4755.64, 4071.98, str(round(100 * (4755.64 - 4071.98) / 4071.98,2)) +
                'mean squared error':[53220620.78, 47576750.63, str(round(100 * (53220620.78 - 49351452.93) / 
                'median absolute error': [2547.79, 953.73, str(round(100 * (2547.79 - 953.73) / 953.73,2)) + '
                'log (MSE)': [0.29, 0.26, str(round(100 * (0.29 - 0.26) / 0.26 ,2)) + ' %']}
for key, value in metrics_dict.items():
    random, optimal, difference = value
    print ("{:<25} {:<15} {:<15} {:<15}".format(key, random, optimal, difference))
```

```
метрика                   случайный       оптимальный     разница, %

mean absolute error       4755.64         4071.98         16.79 %
mean squared error        53220620.78     47576750.63     7.84 %
median absolute error     2547.79         953.73          167.14 %
log (MSE)                 0.29            0.26            11.54 %
```

In [ ]: