**Введение**

Машинное обучение сейчас довольное частое явление в нашей мире. Оно встречается даже там, где мы его не предполагаем увидеть. Целью машинного обучения является решение различного рода задач, самими частыми из которых являются задачи классификации и регрессии. Первые относятся с тем задачам, в которых необходимо узнать класс или вид объекта, как например, определение вида цветка по фотографии. К задачам регрессии же относятся те, в которых необходимо узнать какое-то интервальное значение, как, например, цена квартиры. Методы решения такой задачи и будут рассмотрены далее.

**Цели**

* Собрать данные
* Анализ полученного результата
* Изучение методов регрессионного анализа
* Сравнение методов
* Выбор наилучшего для конкретной задачи

**Сбор данных**

Для сбора данных был выбран сайт cian.ru. Объявления были отфильтрованы по следующим признакам:

* Город Ростов-на-Дону.
* Вторички.
* Были выбраны квартиры с количеством комнат до 4 и студии.

Квартиры типа «студия» были представлены, как квартиры с 0 комнат. Так же наличие таких особенностей как, например, наличие лоджии, обозначаются через 1, а ее оотсутствие через 0.

Для получения html кода страницы воспользуемся скриптом TorCrawler. Его смысл заключается в том, что он отправляет GET запросы через браузер Tor, который использует Proxy сервера. Таким образом нас не будет волновать проблема блокировки IP, так как в этом случае его можно будет заменить его на другой. Для обработки html страницы воспользуемся библиотекой BeautifulSoup.

*Импорт библиотек для обработки html кода*

from TorCrawler import TorCrawler

from bs4 import BeautifulSoup

Все данные будут записывать в csv файл, потому так же подключим библиотеку для работы с ним.

*Импорт библиотеки для работы с csv файла*

import csv

Помимо этого, нам так же понадобиться обрабатывать текстовые данные. Делать это будем с помощью регулярных выражений. Для этого подключим библиотеку для работы с ними.

*Импорт библиотеки для работы со строками*

import re

Объявим объект TorCrawler, с помощью которого будем получать код html страниц.

*Создание объекта TorCrawler*

crawler = TorCrawler(ctrl\_pass='mypassword')

Будем вытаскивать следующие параметры:

1. Комнаты (rooms) – количество комнат
2. Площадь (square) – площадь квартиры
3. Жилая площадь (living space) – жилая площадь квартиры
4. Конкретный этаж (curr\_floor) – этаж, на котором расположена квартира
5. Максимальный этаж (max\_floor) – номер последнего этажа в доме
6. Тип дома (home\_type) – материал стен дома
7. Год постройки (build\_year) – год начала эксплуатации дома
8. Район (district) – район в котором находится дом
9. Грузовой лифт (serv\_lift) – наличие грузового лифта
10. Пассажирский лифт (pass\_lift) – наличие грузового лифта
11. Парковка (parking) – наличие парковки
12. Лоджия (logia) – наличие лоджии
13. Балкон (balcony) – наличие балкона
14. Мусоропровод (garbage\_chute) – наличие мусоропровода
15. Санузел (bathroom) – вид санузла
16. Ремонт (repair\_type) – вид ремонта
17. Вид (wind\_view) – вид из окна
18. Аварийный выход (emergency\_exit) – наличие аварийного выхода
19. Средняя цена за метр (avg\_price\_for\_m) – средняя цена за метр в этом районе
20. Средняя цена за квартиру (avg\_room\_price) – средняя цена за квартиру с соответствующим полю rooms количеством комнат
21. Цена (price) – цена за квартиру

Для начала получим данные с каждой страницы поиска, на которой находятся сами объявления. Ссылка на каждую такую страницу состоит из 2х частей, между которыми вставляется индекс необходимой нам страницы:

*Создание ссылки для получения данных*

base\_url\_1 = 'https://rostov.cian.ru/cat.php?deal\_type=sale&engine\_version=2&max\_house\_year=2019&offer\_type=flat&p='

base\_url\_2 = '&region=4959&room1=1&room2=1&room3=1&room4=1

url = base\_url\_1 + str(i) + base\_url\_2

Где i – это индекс страницы.

Напишем функцию past\_data\_page(url). Она будет отвечать за сбор данных.

Получим html код этой страницы.

*Получение кода html страницы*

soup = crawler.get(url)

Далее найдем html код каждого объявления

*Получение кода каждого объявления*

divs = soup.find('div', class\_ = '\_93444fe79c-wrapper--1Z8Nz')

ads = divs.find\_all('div', class\_ = "\_93444fe79c-card--2Jgih")

Допустим мы хотим получить цену квартиры:

*Получение цены квартиры*

tmp\_url = ads[i].find('div', class\_='c6e8ba5398-info-section--28o47 c6e8ba5398-main-info--Rfnfh').find('a')['href']

tmp\_soup = crawler.get(tmp\_url)

tmp = tmp\_soup.find('div', class\_ = 'a10a3f92e9--container--CKANo').find('span', class\_ = 'a10a3f92e9--price\_value--1iPpd').find('span').text

price = float(re.sub('[(\xa0) ]', '', tmp)[:-1])

Похожие действия производим с остальными параметрами и создаем словарь data вида

*Создание словаря для ввода данных*

data = {'rooms': rooms,

'square': square,

'living\_space': living\_space,

…}

Который и будет записан в csv файл с помощью функции write\_csv(data)

*Создание функции для записи данных в файл*

def write\_csv(data):

with open('apartments\_cian.csv', 'a', encoding='utf-8') as f:

writer = csv.writer(f)

writer.writerow((data['rooms'],

data['square'],

data['living\_space'],

…))

**Визуализация данныхе**

Для визуализации будем использовать библиотеки seaborn и matplotlib

*Импорт библиотек для визуализаии*

import matplotlib.pyplot as plt

import seaborn as sns

Выведем графики зависимости цены от основных параметров.

*Вывод графиков*

colns\_c = 3

fig, axes = plt.subplots(nrows=2, ncols=colns\_c, figsize=(20, 10))

features = ['rooms',

'square',

'living\_space',

'kitchen\_scace',

'build\_year']

for idx, feat in enumerate(features):

sns.scatterplot(x='price', y=feat, data=df, ax=axes[idx // colns\_c, idx % colns\_c])

axes[idx // colns\_c, idx % colns\_c].legend()

axes[idx // colns\_c, idx % colns\_c].set\_xlabel('price')

axes[idx // colns\_c, idx % colns\_c].set\_ylabel(feat)

plt.show()

Интуитивно, можно сделать предположение, что цена будет увеличиваться по мере возрастания площади квартиры. Посмотрим на диаграмму рассеяния для цены и площади:

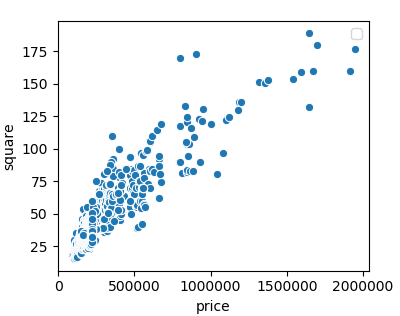


Рис.1 Зависимость цены от площади квартиры

Как можно заметить, они имеют линейную зависимость.

Продолжим исследование с другими параметрами.

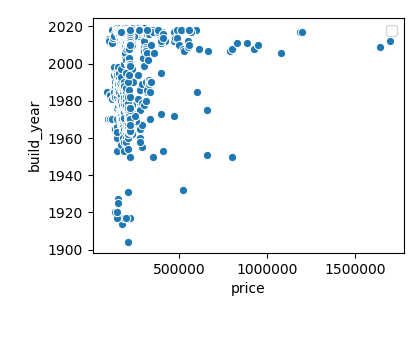


Рис.2 Зависимость цены от года постройки дома

Никакой зависимости здесь не наблюдается.

Графики для жилой площади и площади кухни.

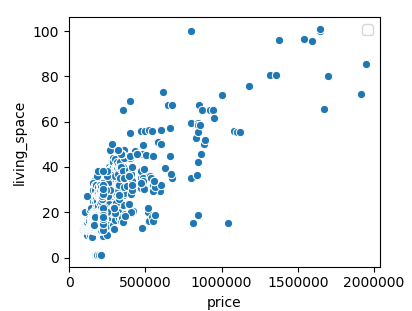


Рис.3 Зависимость цены от жилой площади

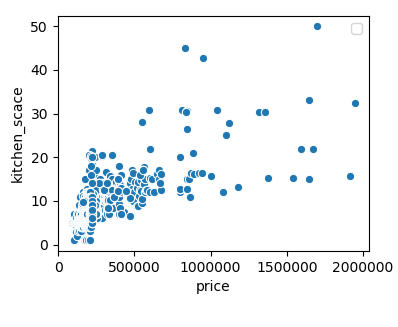


Рис.4 Зависимость цены от площади кухни

Конечно, можно предположить, что эти 2 параметра тоже имеют с ценой линейную зависимость, но это спорное предположение.

Последним интересующем нас параметром остается количество комнат.

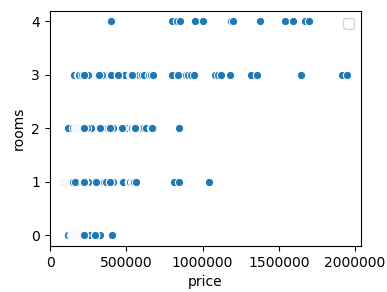


Рис.5 Зависимость цены от площади квартиры

Видно, что на графике присутствуют выбросы, от которых в будущем необходимо избавится.

**Диагностика модели**

Так как по графикам было сделано предположение о линейности рассматриваемой модели, то попробуем сделать предположение, что можно использовать линейную регрессию.

Так как некоторые значения атрибутов содержат строковые значения, то необходимо заменить эти значению на числовые. Для этого разобьем каждый атрибут на несколько по следующему принципу. Например, мы хотим проделать это для атрибута «район». Разбиваем его на атрибуты, количество которых равно количеству уникальных значений в данном атрибуте и, если район квартиры, например, советский, то значение этого атрибута будет равно 1, а все остальные 0. И так проделываем для каждой записи. Сделаем это с помощью функции get\_dummies из библиотеки pandas.

*Импорт библиотеки для работы с БД и разбиение атрибутов*

import pandas as pd

df\_dm = pd.get\_dummies(df, drop\_first=True)

Где df – это наша база данных.

Для того, чтобы модель была корректной необходимо, чтобы она удовлетворяла условиям теоремы Гаусса-Маркова.

1. Отсутствуют пропущенные значения.

Все пропущенные значения были заменены медианами соответствующих атрибутов.

*Заполнение пропущенных значений их медианами*

x\_imp = SimpleImputer(missing\_values=np.nan, strategy="median").fit\_transform(x)

Где x – это изначальные данные.

1. Отсутствует мультиколлинеарность между регрессорами.

Для проверки этого условия необходимо построить матрицу коллинеарности.

*Постороение матрицы корреляции*

corr = df\_dm.corr()

fig, ax = plt.subplots()

ax.matshow(corr, cmap='seismic')

for (i, j), z in np.ndenumerate(corr):

ax.text(j, i, '{:0.1f}'.format(z), ha='center', va='center')

plt.xticks(range(len(corr.columns)), corr.columns, rotation = 90)

plt.yticks(range(len(corr.columns)), corr.columns)

plt.show()

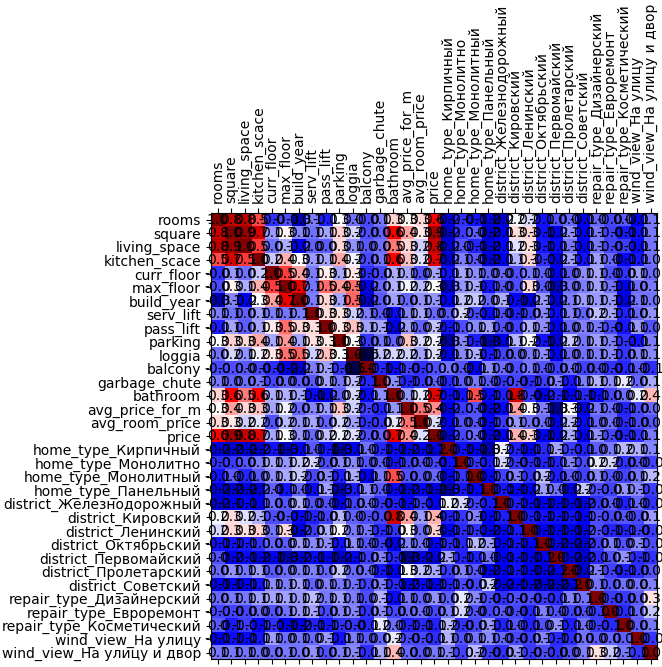


Рис.6 Матрица корреляции

Приблизим самый красный участок.

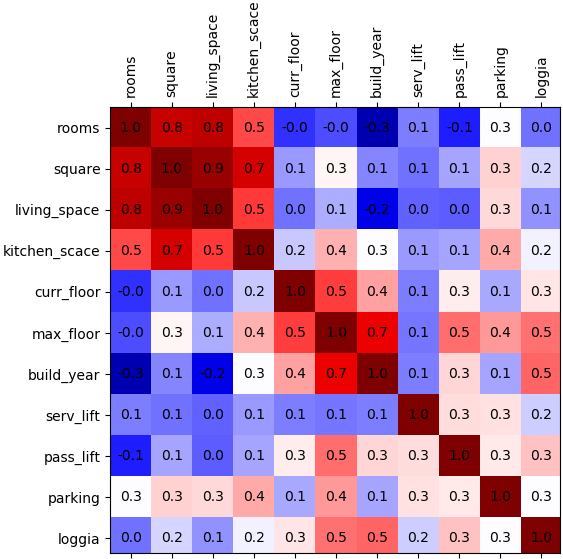


Рис.7 Увеличенная матрица корреляции

Как можно заметить, сильную корреляцию имеют параметры rooms, square и living\_space. (коэффициент корреляции > 0.7). Следовательно, для корректной работы модели необходимо убрать 2 из этих параметров. По графикам можно сделать вывод, что самым ценным для нас будет атрибут square.

1. Отсутствие выбросов.

Выбросы – это наблюдения, выделяющиеся из общей выборки. В нашем случае под ними могут прятаться фейковые квартиры по очень дешевой цене, либо запредельно большой.

Делать это будем с помощью расстояния Кука, которое представляет собой квадратичную зависимость от внутреннего стьюдентизированного остатка, который не рекомендуется использовать для обнаружения выбросов, в отличие от внешнего стьюдентизированного остатка, от которого линейно зависит dffits.

Значимыми для меры dffits являются показатели превосходящие величину 2\*sqrt(k/n) = 0.42, поэтому следует их отбросить (k — количество переменных, n — число строк выборки).

*Удаление выбросов*

m = ols('price ~ square',df).fit()

infl = m.get\_influence()

sm\_fr = infl.summary\_frame()

a = sm\_fr['dffits']<0.2841371938943025

df = df[a]

Таким образом мы убрали 12 значений.

1. Все регрессоры детерминированы и не равны.

Данное условие выполнено.

1. Ошибки не носят систематического характера, дисперсия ошибок одинакова ([гомоскедастичность](https://ru.wikipedia.org/wiki/%D0%93%D0%BE%D0%BC%D0%BE%D1%81%D0%BA%D0%B5%D0%B4%D0%B0%D1%81%D1%82%D0%B8%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C))

Для расчета ошибок воспользуемся методом resid.

*Расчет ошибок*

m = ols('price ~ square',df).fit().resid

Выведем графики распределения ошибок от нескольких переменных.

*Вывод графики распределения ошибок от площади квартиры*

sns.scatterplot(x='square’, y=m, data=df)

plt.show()

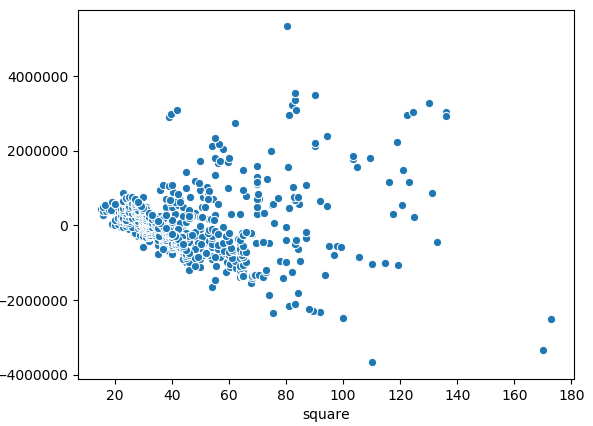


Рис.7 График распределения ошибок от площади квартиры

*Вывод графики распределения ошибок от количества комнат*

*квартиры*

sns.scatterplot(x='rooms', y=m, data=df)

plt.show()

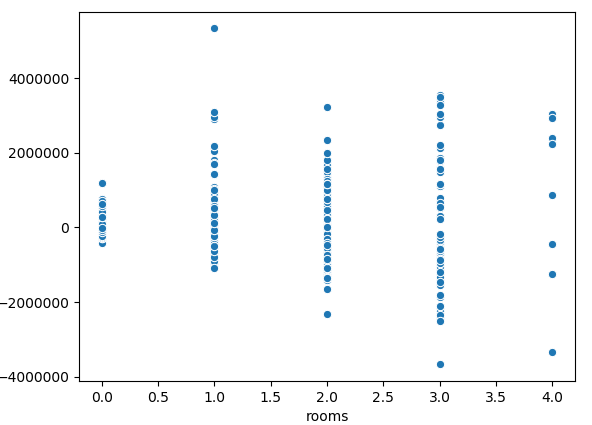


Рис.8 Вывод графики распределения ошибок от площади квартиры

*Вывод графики распределения ошибок от площади квартиры*

sns.scatterplot(x='max\_floor', y=m, data=df)

plt.show()

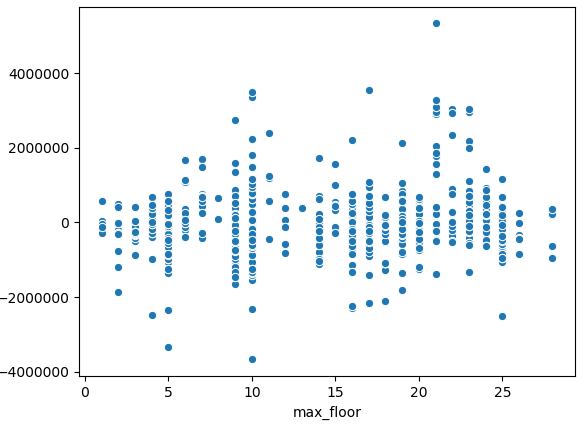


Рис.9 Вывод графики распределения ошибок от количества этажей в доме

Ошибки распределены равномерно относительно горизонтальной оси, что подтверждает выполнение условия гомоскедастичности.

1. Ошибки распределены нормально.

Для проверки данного условия построим график квантилей остатков против квантилей, которые можно было бы ожидать при условии, что остатки нормально распределены.

*Вывод графика квантилей остатков*

import pylab

sm.qqplot(m, line='s')

pylab.show()

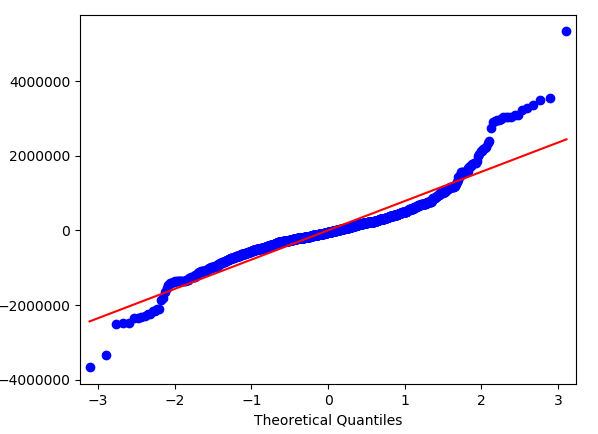


Рис.10 График квантилей остатков

**Сравнение моделей**

Для решения нашей задачи воспользуемся несколькими моделями машинного обучения:

1. Множественная линейная регрессия
2. Ридж-регрессия
3. Случайный лес

Линейная регрессия – это модель, которая объясняет зависимость одной переменной Y от другой или нескольких других переменных X с линейной функцией зависимости. Она не умеет отбирать лучшие атрибуты для предсказания. Их приходится выбирать вручную. Из-за этого может происходить перетренированность модели, что приведет к тому, что на тестовом наборе данных результаты могут намного хуже ожидаемых.

Воспользуемся готовой моделью линейной регрессии из библиотеки sklearn:

*Импорт линейной регрессии*

from sklearn.linear\_model import LinearRegression

Разобьем нашу базу на тренировочный и тестовый набор данных для того, чтобы проверить, как хорошо модель справляется с задачей. Сделаем это с помощью функции train\_test\_split из библиотеки sklearn:

*Разбиение данных на тренировочные и тестовые*

from sklearn.model\_selection import train\_test\_split

x = df.drop('price', 1)

y = df['price']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.45, random\_state=24)

Теперь можно «тренировать» модель с помощью метода fit():

*Тренировка модели*

model = LinearRegression().fit(x\_train, y\_train)

Для проверки качественности выполненной работы воспользуемся коэффициентом детерминации. Получить его можно с помощью метода score():

*Получение коэффициента детерминации для линейной регрессии*

r2\_train = model.score(x\_train, y\_train)

r2\_test = model.score(x\_test, y\_test)

print("Train squared: {}".format(r21))

print("Tuned squared: {}".format(r2))

На обучающей выборке получили значение 0.8416767613225101, что свидетельствует о том, что модель не перетренирована. Значение коэффициента детерминации на тестовой выборке равно 0.8407216669327404, что является довольно неплохим результатом.

Перейдем к следующей модели – Ридж-регрессии. Это один из методов, который применяют для борьбы с переизбытком данных. Он так же является линейной моделью. Принцип работы модели заключается в том, что коэффициенты линейной регрессии оставались как можно меньше, но качество предсказания не ухудшалось.

Импортируем модель из библиотеки sklearn:

*Импорт ридж-регрессии*

from sklearn.linear\_model import Ridge

За регуляцию коэффициентов отвечает гиперпараметр Alpha. Чем точнее он будет подобран, тем лучше будет работать наша модель. Для того, чтобы получить наилучшее его значение воспользуемся методом Перекрестной проверки. Импортируем его из библиотеки sklearn и найдем лучшее значение:

*Получение коэффициента детерминации для ридж-регрессии*

from sklearn.model\_selection import GridSearchCV

gm\_cv = GridSearchCV(Ridge(), {'alpha': np.logspace(-5,8,15)}, cv = 5)

print("Tuned params: {}".format(gm\_cv.best\_params\_))

Получили значение 3.727593720314938. Далее также натренируем модель и узнаем коэффициент детерминации для нее:

*Тренировка ридж-регрессии и получение коэффициента детерминации*

gm\_cv.fit(x\_train, y\_train)

r2\_train = gm\_cv.score(x\_train, y\_train)

r2\_test = gm\_cv.score(x\_test, y\_test)

Для тренировочного набора получили значение 0.8398932584122097, а для тестового 0.8382245142829319. Значения получились почти равными по сравнению с обычной линейной регрессией. Это значит, что первая модель не была перетренирована.

Рассмотрим последнюю модель регрессионного анализа «Случайный лес». Она уже не является линейной и работает по другому принципу. Для начала разберем 2 метода, которые лежат в основе данной модели: Бутсрэп и Бэггинг.

Работа Бутстрэпа заключается в следующем. Из выборки Х размера N берется N объектов с возвращением. Полученную выборку обозначим X1. В ней могут одинаковые элементы. Повторяем процедуру M раз и генерируем M подвыборок Х1 … ХM. С достаточно большим количеством выборок можно оценивать различные статистики исходного распределения.

Зная принцип работы Бутстрэпа, можно перейти к Бэеггингу. Разобьем выборку Х с помощью Бутстрепа на Х1 … ХM и на каждой выборке обучим регрессор. В итоге усредним ответы этих алгоритмов и таким образом получим итоговый регрессор.

Алгоритм построения случайного леса, состоящего из N деревьев, выглядит следующим образом:

* Для каждого n=1,…,N:
  + Сгенерировать выборку Xn с помощью бутстрэпа;
  + Построить решающее дерево bn по выборке Xn:  
    — по заданному критерию мы выбираем лучший признак, делаем разбиение в дереве по нему и так до исчерпания выборки  
    — дерево строится, пока в каждом листе не более nmin объектов или пока не достигнем определенной высоты дерева  
    — при каждом разбиении сначала выбирается m случайных признаков из n исходных,  
    и оптимальное разделение выборки ищется только среди них.

Импортируем Случайный лес из библиотеки sklearn:

*Импорт случайного леса*

from sklearn.ensemble import RandomForestRegressor

Так же настроем гиперпараметры для для этой модели:

*Поиск и вывод лучшего гиперпараметра*

tree = RandomForestRegressor(n\_estimators=100, n\_jobs=-1, random\_state=17)

gm\_cv = GridSearchCV(tree, {'max\_depth': range(9,15)}, cv = 5)

print(gm\_cv.best\_params\_)

Получили значение параметра max\_depth = 10. Это означает, что глубина каждого дерева не будет превышать 10.

Натренеруем модель и узнаем коэффициент детерминации:

*Тренировка и вывод коэффициента детерминации для случайного леса*

gm\_cv.fit(x\_train, y\_train)

r2\_train = gm\_cv.score(x\_train, y\_train)

r2\_test = gm\_cv.score(x\_test, y\_test)

На тренировочном наборе получаем значение 0.9763162815147894, а на тестовом 0.9050745714428732. Случайный лес отработал в разы лучше, чем линейные регрессии, но это не означает, что эта модель всегда будет превосходить по результатам остальные.

**Плюсы и минусы**

Линейные алгоритмы

Плюсы

* Очень быстрые, могут работать на очень больших выборках
* Практически вне конкуренции, когда признаков очень много (от сотен тысяч и более), и они разреженные (хотя есть еще факторизационные машины)
* Коэффициенты перед признаками могут интерпретироваться (при условии что признаки масштабированы) – в линейной регрессии как частные производные зависимой переменной от признаков, в логистической – как изменение шансов на отнесение к одному из классов в expβi раз при изменении признака xi на 1 ед.
* Логистическая регрессия выдает вероятности отнесения к разным классам (это очень ценится, например, в кредитном скоринге)
* Модель может строить и нелинейную границу, если на вход подать полиномиальные признаки

Минусы:

* Плохо работают в задачах, в которых зависимость ответов от признаков сложная, нелинейная
* На практике предположения теоремы Маркова-Гаусса почти никогда не выполняются, поэтому чаще линейные методы работают хуже, чем, например, SVM и ансамбли (по качеству решения задачи классификации/регрессии)

Плюсы и минусы случайного леса

**Плюсы**:  
— имеет высокую точность предсказания, на большинстве задач будет лучше линейных алгоритмов; точность сравнима с точностью бустинга  
— практически не чувствителен к выбросам в данных из-за случайного сэмлирования  
— не чувствителен к масштабированию (и вообще к любым монотонным преобразованиям) значений признаков, связано с выбором случайных подпространств  
— не требует тщательной настройки параметров, хорошо работает «из коробки». С помощью «тюнинга» параметров можно достичь прироста от 0.5 до 3% точности в зависимости от задачи и данных  
— способен эффективно обрабатывать данные с большим числом признаков и классов  
— одинаково хорошо обрабатывет как непрерывные, так и дискретные признаки  
— редко переобучается, на практике добавление деревьев почти всегда только улучшает композицию, но на валидации, после достижения определенного количества деревьев, кривая обучения выходит на асимптоту  
— для случайного леса существуют методы оценивания значимости отдельных признаков в модели  
— хорошо работает с пропущенными данными; сохраняет хорошую точность, если большая часть данных пропущенна  
— предполагает возможность сбалансировать вес каждого класса на всей выборке, либо на подвыборке каждого дерева  
— вычисляет близость между парами объектов, которые могут использоваться при кластеризации, обнаружении выбросов или (путем масштабирования) дают интересные представления данных  
— возможности, описанные выше, могут быть расширены до неразмеченных данных, что приводит к возможности делать кластеризацию и визуализацию данных, обнаруживать выбросы  
— высокая параллелизуемость и масштабируемость.

**Минусы**:  
— в отличие от одного дерева, результаты случайного леса сложнее интерпретировать  
— нет формальных выводов (p-values), доступных для оценки важности переменных  
— алгоритм работает хуже многих линейных методов, когда в выборке очень много разреженных признаков (тексты, Bag of words)  
— случайный лес не умеет экстраполировать, в отличие от той же линейной регрессии (но это можно считать и плюсом, так как не будет экстремальных значений в случае попадания выброса)  
— алгоритм склонен к переобучению на некоторых задачах, особенно на зашумленных данных  
— для данных, включающих категориальные переменные с различным количеством уровней, случайные леса предвзяты в пользу признаков с большим количеством уровней: когда у признака много уровней, дерево будет сильнее подстраиваться именно под эти признаки, так как на них можно получить более высокое значение оптимизируемого функционала (типа прироста информации)  
— если данные содержат группы коррелированных признаков, имеющих схожую значимость для меток, то предпочтение отдается небольшим группам перед большими  
— больший размер получающихся моделей. Требуется O(NK) памяти для хранения модели, где K — число деревьев.

**Заключение**

В результате проведенного исследования был проанализирован рынок недвижимости города Ростов-на-Дону. Полученные данные позволили выявить атрибуты квартир, которые сильнее всего влияют на их цены. Ими оказались: площадь квартиры, жилая площадь, площадь кухни.

Так же на основе этих данных были изучены методы регрессионного анализа. В качестве моделей использовались: линейная регрессия, ридж-регрессия, случайный лес. Каждая модель обучилась на предоставленных ей данных и после было проведено сравнение их между собой. Для каждой модели так же высчитан коэффициент детерминации на тренировочном и тестовом наборе данных. Кроме того, были выявлены их плюсы и минусы.

В результате сравнения выяснилось, что наилучшей моделью конкретно для данной задачи оказался случайный лес.

**Список литературы**

1. scikit-learn: machine learning in Python — scikit-learn 0.21.2 documentation. — URL: http://scikit-learn.org/stable/ (дата обр. 01.05.2019)
2. Pandas 0.24.2 documentation – URL: <http://pandas.pydata.org/pandas-docs/stable/> (дата обр. 01.05.2019)
3. Случайный лес. – URL: https://habr.com/ru/company/ods/blog/324402/#2-sluchaynyy-les/ (дата обр. 01.05.2019)
4. Как программист машину покупал. – URL: https://habr.com/ru/post/302788/ (дата обр. 01.05.2019)