Лабораторная работа №5

Изучение моделей классификации и регрессии.

Оглавление

[Ход работы: 1](#_Toc149215902)

[Построение моделей классификации и регрессии 1](#_Toc149215903)

[Подключение библиотек 3](#_Toc149215904)

[Создание или подключение датасетов 3](#_Toc149215905)

[Метод k-ближайших соседей 4](#_Toc149215906)

[Классификация методом k-ближайших соседей 5](#_Toc149215907)

[Оценка модели 8](#_Toc149215908)

[Анализ KNeighborsClassifier 8](#_Toc149215909)

[Регрессия ближайших соседей 11](#_Toc149215910)

[Прогнозы модели регрессии 12](#_Toc149215911)

[Анализ KNeighborsRegressor 13](#_Toc149215912)

[Линейные модели регрессии 14](#_Toc149215913)

[Линейная модель регрессии (метод наименьших квадратов) 16](#_Toc149215914)

[Гребневая регрессия 17](#_Toc149215915)

[Лассо 22](#_Toc149215916)

[Линейные модели для классификации 24](#_Toc149215917)

[Построение модели LogisticRegression 26](#_Toc149215918)

[Задание: 28](#_Toc149215919)

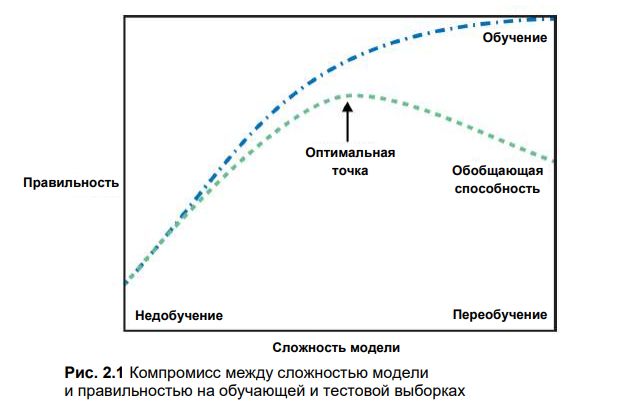
**Цель работы**: Применить библиотеки sсikit-learn и mglearn для построения моделей классификации и регрессии. Изучить полученные данные, научиться оценивать качество моделей.

# Ход работы:

## Построение моделей классификации и регрессии

При построении модели возможны варианты:

* Переобучение (модель подстраивается под новые данные)
* Недообучение (модель плохо работает даже на обучающем наборе)
* Оптимальная точка (дает наилучшую обобщающую способность)



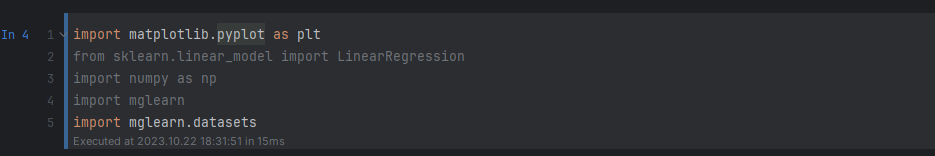
*Рис. 1а Компромисс между сложностью модели и правильностью на обучающей и тестовой выборках*

Цель регрессии состоит в том, чтобы спрогнозировать непрерывное число или число с плавающей точкой (floating-point number), если использовать термины программирования, или вещественное число (real number), если говорить языком математических терминов.

Прогнозирование годового дохода человека в зависимости от его образования, возраста и места жительства является примером регрессионной задачи. Прогнозируемое значение дохода представляет собой сумму (amount) и может быть любым числом в заданном диапазоне. Другой пример регрессионной задачи – прогнозирование объема урожая зерна на ферме в зависимости от таких атрибутов, как объем предыдущего урожая, погода, и количество сотрудников, работающих на ферме. И снова объем урожая может быть любым числом.

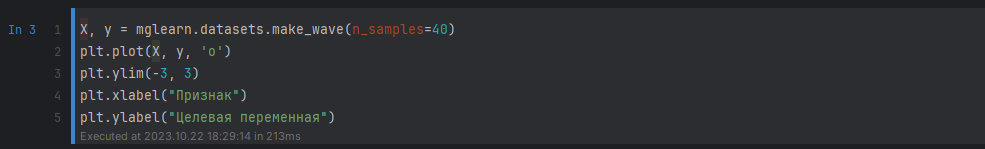
Для решения задач классификации и регрессии необходимо подключить следующие библиотеки.

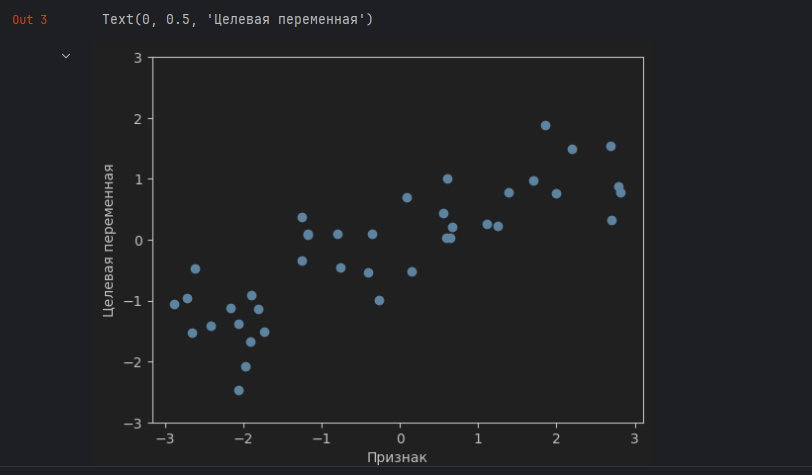
### Подключение библиотек



### Создание или подключение датасетов

Для иллюстрации алгоритмов регрессии, воспользуемся синтетическим набором **wave**. Набор данных имеет единственный входной признак и непрерывную целевую переменную или отклик (response), который хотим смоделировать. На рисунке, построенном ниже (рис. 1), по оси x располагается единственный признак, а по оси y – целевая переменная (ответ).

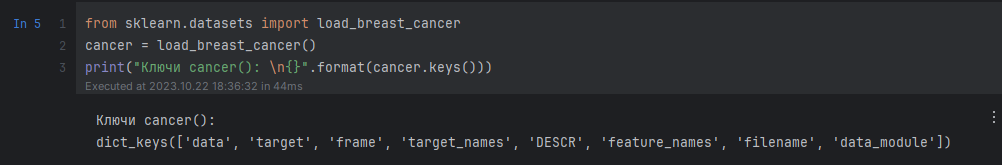


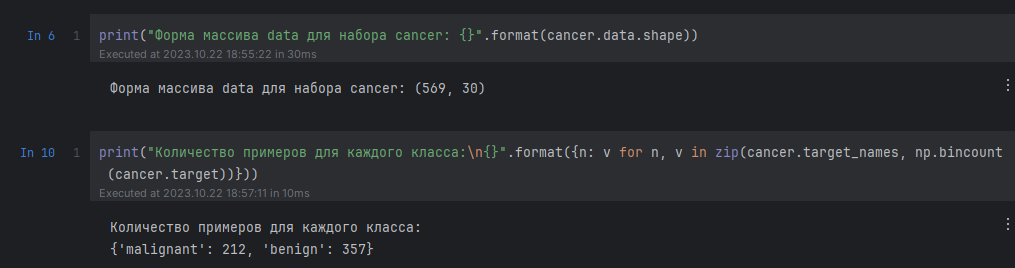


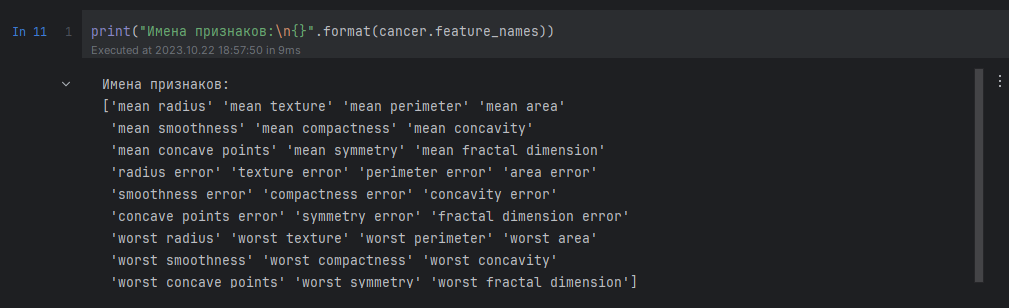
*Рис. 1 График для набора данных wave, по оси x отложен признак, по оси у – целевая переменная*

Дополним данные примерами демо датасетов

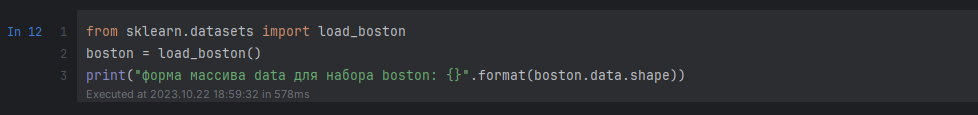
**Датасет (load\_breast\_cancer)**

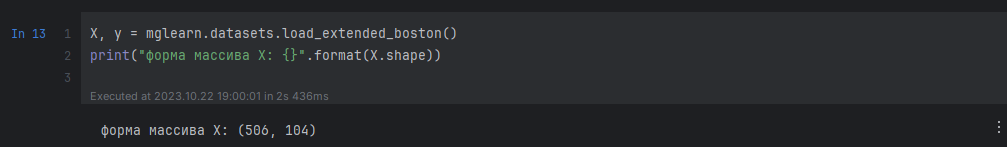






**Датасет (load\_boston)**



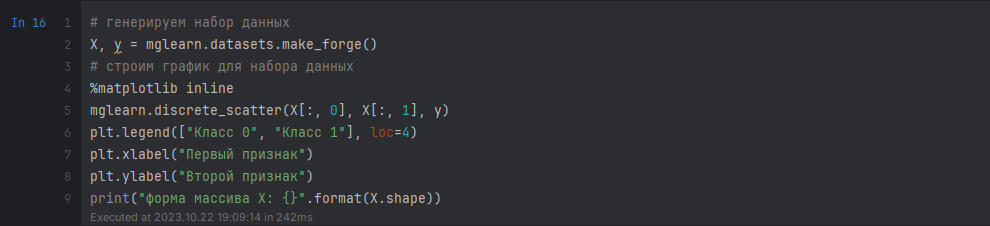


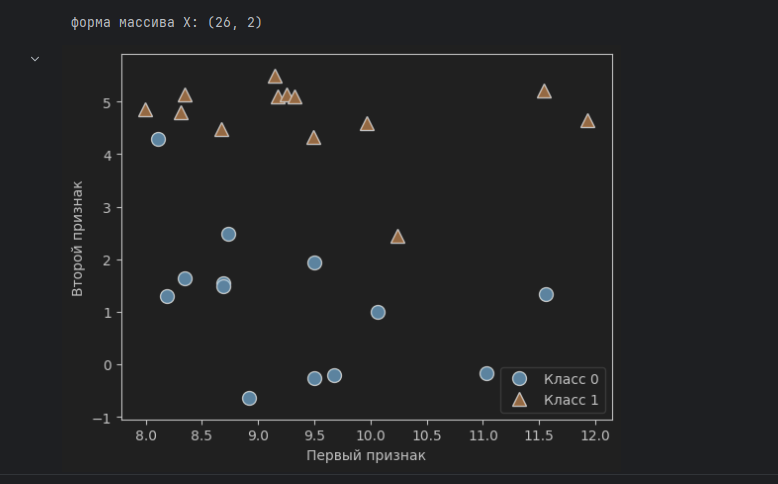
## Метод k-ближайших соседей

Алгоритм k ближайших соседей, возможно, является самым простым алгоритмом машинного обучения. Построение модели заключается в запоминании обучающего набора данных. Для того, чтобы сделать прогноз для новой точки данных, алгоритм находит ближайшие к ней точки обучающего набора, то есть находит «ближайших соседей».

### Классификация методом k-ближайших соседей

В простейшем варианте алгоритм k-ближайших соседей рассматривает лишь одного ближайшего соседа – точку обучающего набора, ближе всего расположенную к точке, для которой мы хотим получить прогноз. Прогнозом является ответ, уже известный для данной точки обучающего набора. На рис. 2а, 2 показано решение задачи классификации для набора данных forge:



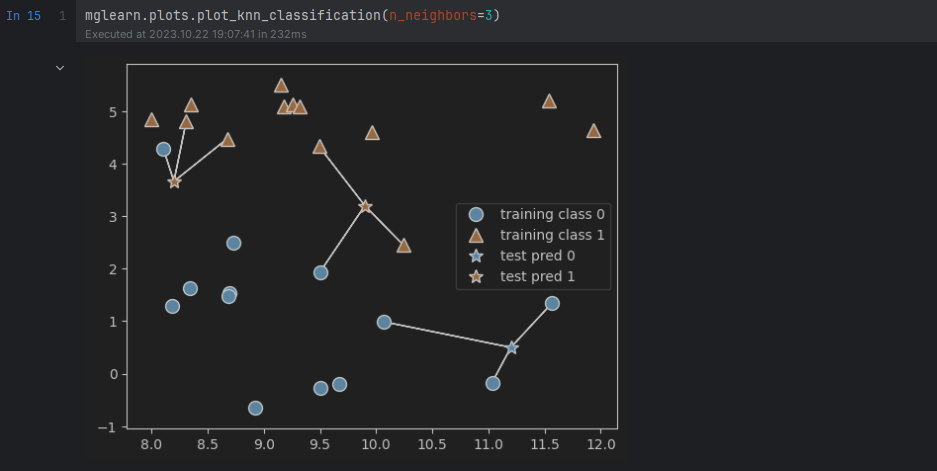


*Рис. 2а Диаграмма рассеяния для набора данных forge*



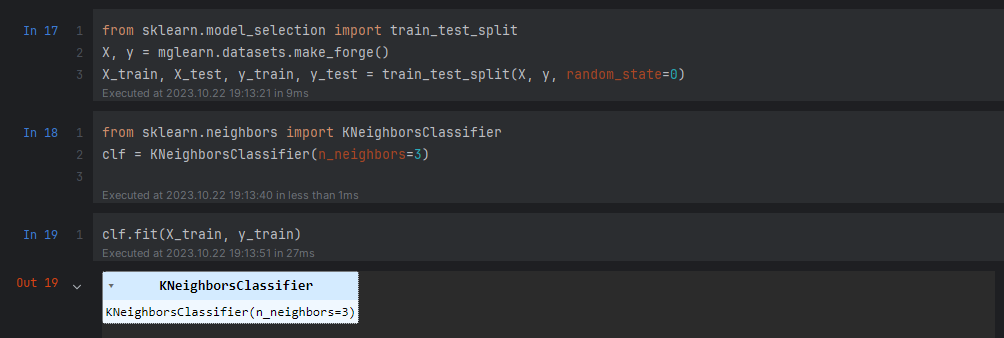
*Рис. 2 Прогнозы, полученные для набора данных forge с помощью модели одного ближайшего соседа*

Здесь добавили три новые точки данных, показанные в виде звездочек. Для каждой отметили ближайшую точку обучающего набора. Прогноз, который дает алгоритм одного ближайшего соседа – метка этой точки (показана цветом маркера). Вместо того, чтобы учитывать лишь одного ближайшего соседа, можем рассмотреть произвольное количество (k) соседей. Отсюда и происходит название алгоритма k ближайших соседей. Когда рассматриваем более одного соседа, для присвоения метки используется голосование (voting). Это означает, что для каждой точки тестового набора подсчитывается количество соседей, относящихся к классу 0, и количество соседей, относящихся к классу 1. Затем присваивается точке тестового набора наиболее часто встречающийся класс: другими словами, выбирается класс, набравший большинство среди k ближайших соседей. В примере, приведенном ниже (рис. 3), используются три ближайших соседа:

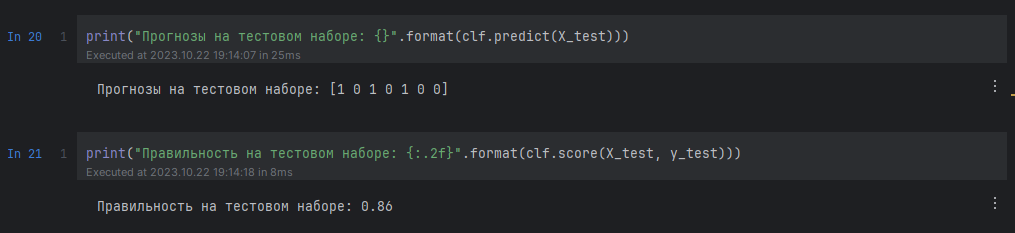


*Рис. 3 Прогнозы, полученные для набора данных forge с помощью модели трех ближайших соседей*

И снова прогнозы переданы цветом маркера. Видно, что прогноз для новой точки данных в верхнем левом углу отличается от прогноза, полученного при использовании одного ближайшего соседа. Хотя данный рисунок иллюстрирует задачу бинарной классификации, этот метод можно применить к наборам данных с любым количеством классов. В случае мультиклассовой классификации подсчитываем количество соседей, принадлежащих к каждому классу, и снова прогнозируем наиболее часто встречающийся класс. Теперь давайте посмотрим, как можно применить алгоритм k ближайших соседей, используя scikit-learn. Во-первых, разделим наши данные на обучающий и тестовый наборы, чтобы оценить обобщающую способность модели:

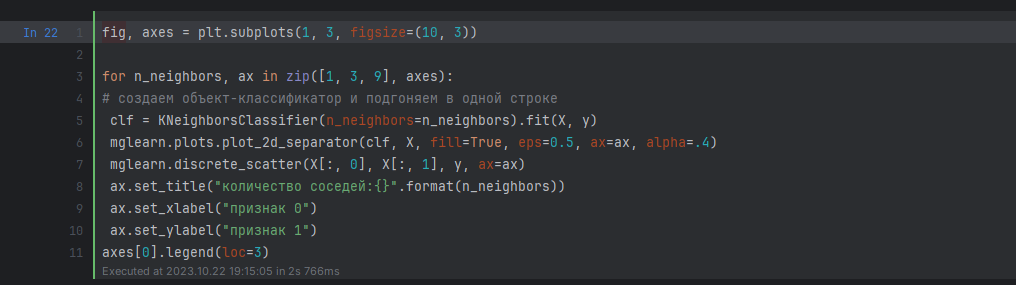


### Оценка модели



### Анализ KNeighborsClassifier

Кроме того, для двумерных массивов данных можем показать прогнозы для всех возможных точек тестового набора, разместив в плоскости ху. Зададим цвет плоскости в соответствии с тем классом, который будет присвоен точке в этой области. Это позволит сформировать границу принятия решений (decision boundary), которая разбивает плоскость на две области: область, где алгоритм присваивает класс 0, и область, где алгоритм присваивает класс 1. Программный код, приведенный ниже, визуализирует границы принятия решений для одного, трех и девяти соседей (показаны на рис. 4):



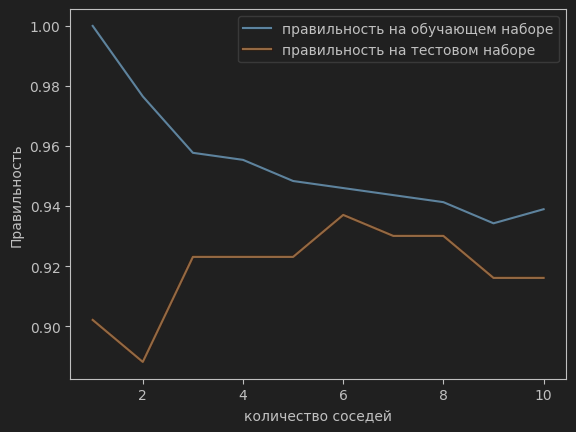


*Рис. 4 Границы принятия решений, созданные моделью ближайших соседей для различных значений n\_neighbors*

На рисунке слева можно увидеть, что использование модели одного ближайшего соседа дает границу принятия решений, которая очень хорошо согласуется с обучающими данными. Увеличение числа соседей приводит к сглаживанию границы принятия решений. Более гладкая граница соответствует более простой модели. Другими словами, использование нескольких соседей соответствует высокой сложности модели, а использование большого количества соседей соответствует низкой сложности модели.

Если взять крайний случай, когда количество соседей будет равно количеству точек данных обучающего набора, каждая точка тестового набора будет иметь одних и тех же соседей (соседями будет все точки обучающего набора) и все прогнозы будут одинаковыми: будет выбран класс, который является наиболее часто встречающимся в обучающем наборе. Выясним, существует ли взаимосвязь между сложностью модели и обобщающей способностью, о которой говорили ранее. Сделаем это с помощью реального набора данных **Breast Cancer**. Начнем с того, что разобьем данные на обучающий и тестовый наборы. Затем оценим качество работы модели на обучающем и тестовом наборах с использованием разного количества соседей. Результаты показаны на рис. 5:

from sklearn.datasets import load\_breast\_cancer  
cancer = load\_breast\_cancer()  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 cancer.data, cancer.target, stratify=cancer.target, random\_state=66)  
training\_accuracy = []  
test\_accuracy = []  
# пробуем n\_neighbors от 1 до 10  
neighbors\_settings = range(1, 11)  
  
for n\_neighbors in neighbors\_settings:  
 # строим модель  
 clf = KNeighborsClassifier(n\_neighbors=n\_neighbors)  
 clf.fit(X\_train, y\_train)  
 # записываем правильность на обучающем наборе  
 training\_accuracy.append(clf.score(X\_train, y\_train))  
 # записываем правильность на тестовом наборе  
 test\_accuracy.append(clf.score(X\_test, y\_test))  
plt.plot(neighbors\_settings, training\_accuracy, label="правильность на обучающем наборе")  
plt.plot(neighbors\_settings, test\_accuracy, label="правильность на тестовом наборе")  
plt.ylabel("Правильность")  
plt.xlabel("количество соседей")  
plt.legend()

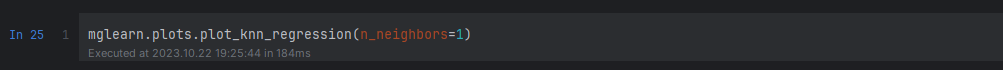


*Рис. 5. Сравнение правильности на обучающем и тестовом наборах как функции от количества соседей*

На этом графике по оси y отложена правильность на обучающем наборе и правильность на тестовом наборе, а по оси x – количество соседей. В реальности подобные графики редко бывают гладкими, мы по-прежнему можем увидеть некоторые признаки переобучения и недообучения (обратите внимание, что поскольку использование небольшого количества соседей соответствует более сложной модели, график представляет собой изображение рис. 1а, зеркально отраженное по горизонтали). При использовании модели одного ближайшего соседа правильность на обучающем наборе идеальна. Однако при использовании большего количества соседей модель становится все проще и правильность на обучающем наборе падает. Правильность на тестовом наборе в случае использования одного соседа ниже, чем при использовании нескольких соседей. Это указывает на то, что использование одного ближайшего соседа приводит к построению слишком сложной модели. С другой стороны, когда используются 10 соседей, модель становится слишком простой и она работает еще хуже. Оптимальное качество работы модели наблюдается где-то посередине, когда используются шесть соседей. Однако посмотрим на шкалу y. Худшая по качеству модель дает правильность на тестовом наборе около 88%, что по-прежнему может быть приемлемым результатом.

## Регрессия ближайших соседей

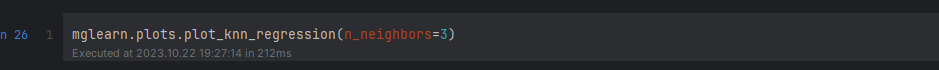
Существует также регрессионный вариант алгоритма k ближайших соседей. Опять же, давайте начнем с рассмотрения одного ближайшего соседа, на этот раз воспользуемся набором данных wave. Мы добавили три точки тестового набора в виде зеленых звездочек по оси х. Прогноз с использованием одного соседа – это целевое значение ближайшего соседа. На рис. 6 прогнозы показаны в виде синих звездочек:

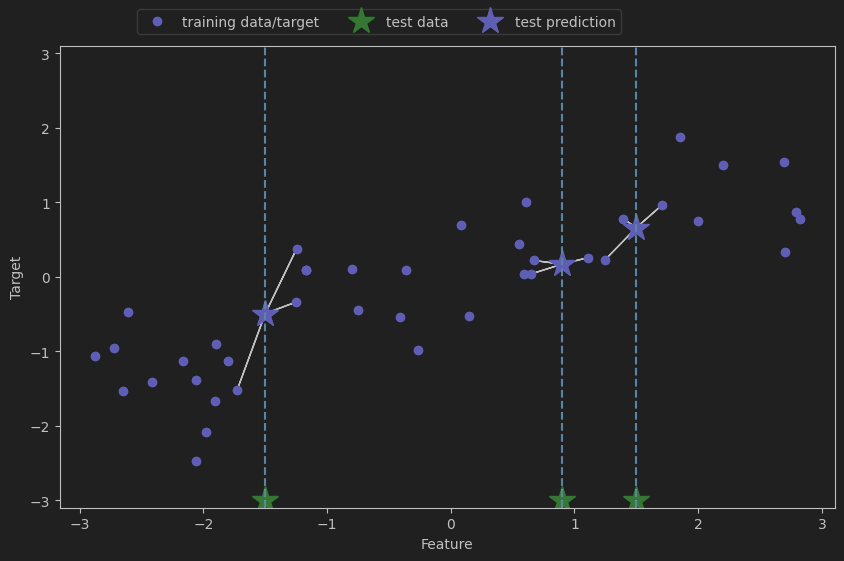




*Рис. 6 Прогнозы, полученные с помощью регрессионной модели одного ближайшего соседа для набора данных wave*

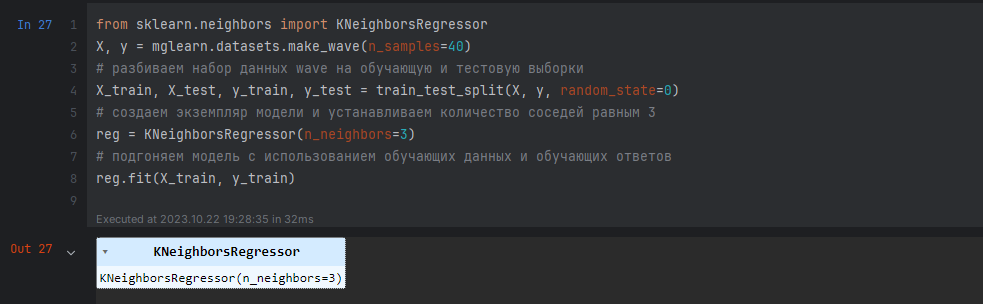
И снова для регрессии мы можем использовать большее количество ближайших соседей. При использовании нескольких ближайших соседей прогнозом становится среднее значение соответствующих соседей (рис. 7):



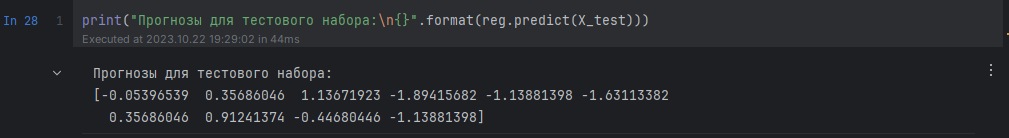


*Рис. 7 Прогнозы, полученные с помощью регрессионной модели трех ближайших соседей для набора данных wave*

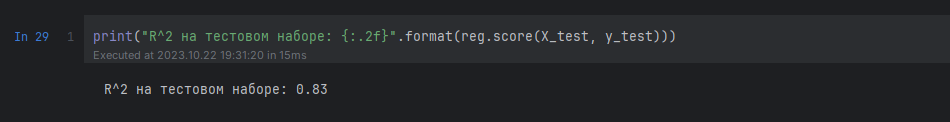
Алгоритм регрессии k ближайших соседей реализован в классе KNeighborsRegressor. Он используется точно так же, как KNeighborsClassifier:



### Прогнозы модели регрессии



Кроме того, мы можем оценить качество модели с помощью метода score, который для регрессионных моделей возвращает значение R2. R2, также известный как коэффициент детерминации, является показателем качества регрессионной модели и принимает значения от 0 до 1. Значение 1 соответствует идеальной прогнозирующей способности, а значение 0 соответствует константе модели, которая лишь предсказывает среднее значение ответов в обучающем наборе, y\_train:

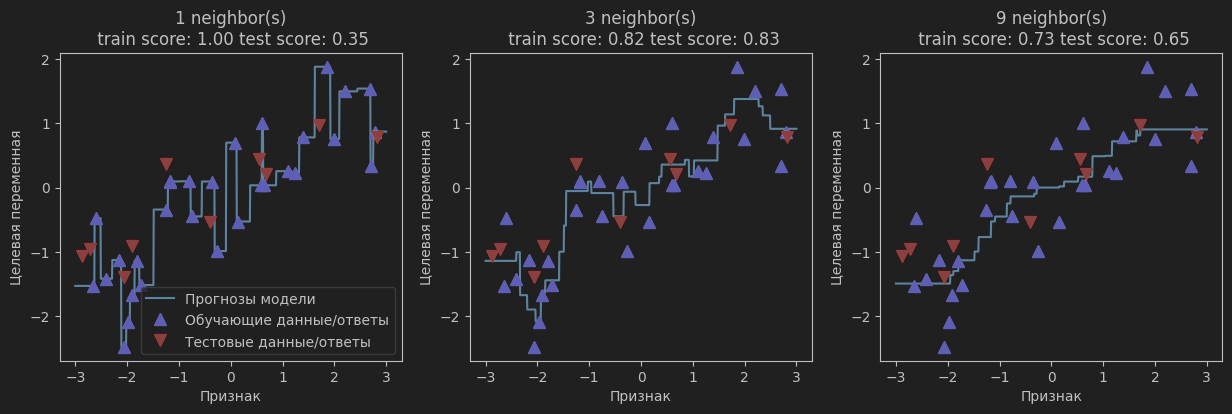


В данном случае значение R 2 составляет 0.83, что указывает на относительно хорошее качество подгонки модели.

### Анализ KNeighborsRegressor

Применительно к нашему одномерному массиву данных мы можем увидеть прогнозы для всех возможных значений признаков (рис. 8). Для этого мы создаем тестовый набор данных и визуализируем полученные линии прогнозов:

fig, axes = plt.subplots(1, 3, figsize=(15, 4))  
# создаем 1000 точек данных, равномерно распределенных между -3 и 3  
line = np.linspace(-3, 3, 1000).reshape(-1, 1)  
for n\_neighbors, ax in zip([1, 3, 9], axes):  
 # получаем прогнозы, используя 1, 3, и 9 соседей  
 reg = KNeighborsRegressor(n\_neighbors=n\_neighbors)  
 reg.fit(X\_train, y\_train)  
 ax.plot(line, reg.predict(line))  
 ax.plot(X\_train, y\_train, '^', c=mglearn.cm2(0), markersize=8)  
 ax.plot(X\_test, y\_test, 'v', c=mglearn.cm2(1), markersize=8)  
  
 ax.set\_title(  
 "{} neighbor(s)\n train score: {:.2f} test score: {:.2f}".format(  
 n\_neighbors, reg.score(X\_train, y\_train),  
 reg.score(X\_test, y\_test)))  
 ax.set\_xlabel("Признак")  
 ax.set\_ylabel("Целевая переменная")  
axes[0].legend(["Прогнозы модели", "Обучающие данные/ответы",  
 "Тестовые данные/ответы"], loc="best")



*Рис. 8 Сравнение прогнозов, полученных с помощью регрессии ближайших соседей для различных значений n\_neighbors*

Как видно на графике, при использовании лишь одного соседа каждая точка обучающего набора имеет очевидное влияние на прогнозы, и предсказанные значения проходят через все точки данных. Это приводит к очень неустойчивым прогнозам. Увеличение числа соседей приводит к получению более сглаженных прогнозов, но при этом снижается правильность подгонки к обучающим данным.

## Линейные модели регрессии

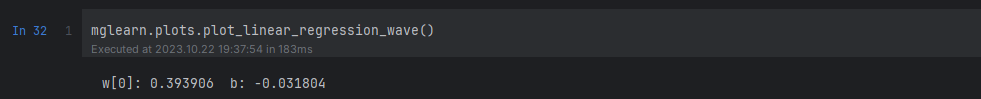
Для регрессии общая прогнозная формула линейной модели выглядит следующим образом:



Здесь х[0] по х[р] обозначают признаки (в данном примере число характеристик равно р+1) для отдельной точки данных, w и b – параметры модели, оцениваемые в ходе обучения, и y ˆ – прогноз, выдаваемый моделью. Для набора данных с одним признаком эта формула имеет вид:



Возможно, из школьного курса математики вы вспомните, что эта формула – уравнение прямой. Здесь х[0] является наклоном, а b – сдвигом по оси y. Когда используется несколько признаков, регрессионное уравнение содержит параметры наклона для каждого признака. Как вариант, прогнозируемый ответ можно представить в виде взвешенной суммы входных признаков, где веса (которые могут быть отрицательными) задаются элементами w. Попробуем вычислить параметры w[0] и b для нашего одномерного набора данных wave с помощью следующей строки программного кода (см. рис. 9):



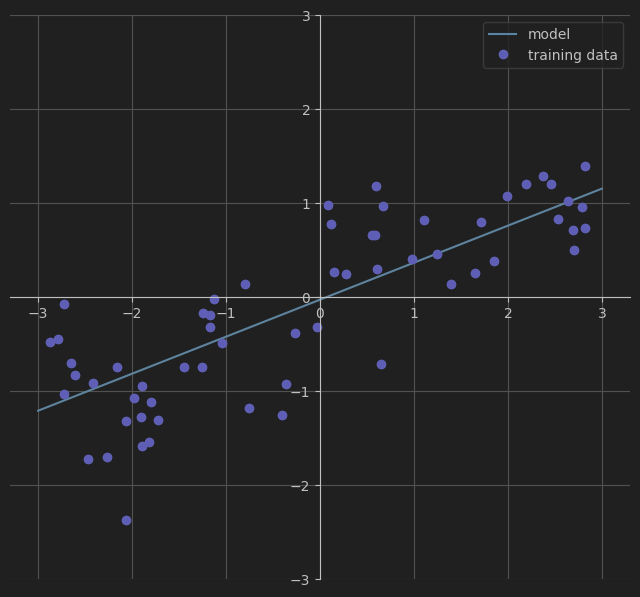
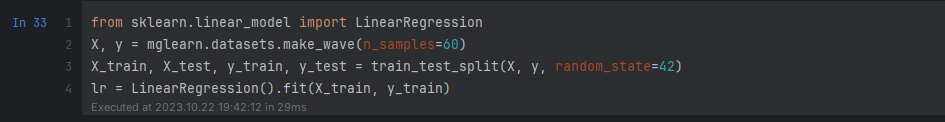


Рис. 9 Прогнозы линейной модели для набора данных wave

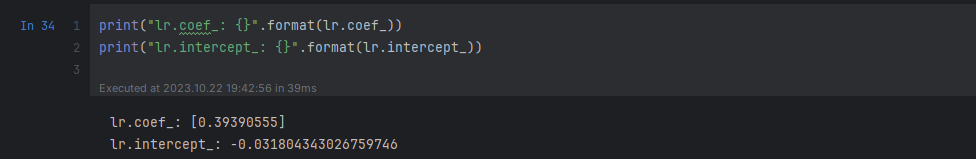
Линейные модели для регрессии можно охарактеризовать как регрессионные модели, в которых прогнозом является прямая линия для одного признака, плоскость, когда используем два признака, или гиперплоскость для большего количества измерений (то есть, когда используем много признаков). Если прогнозы, полученные с помощью прямой линии, сравнить с прогнозами KNeighborsRegressor (рис. 9), использование линии регрессии для получения прогнозов кажется очень строгим. Похоже, что все мелкие детали данных не учитываются. В некотором смысле это верно. Мы видвигаем сильное (и в некоторой степени нереальное) предположение, что наша целевая переменная у является линейной комбинацией признаков. Однако анализ одномерных данных дает несколько искаженную картину. Для наборов данных с большим количеством признаков линейные модели могут быть очень полезны. В частности, если у вас количество признаков превышает количество точек данных для обучения, любую целевую переменную у можно прекрасно смоделировать (на обучающей выборке) в виде линейной функции. Существует различные виды линейных моделей для регрессии. Различие между этими моделями заключается в способе оценивания параметров модели w и b по обучающим данным и контроле сложности модели. Теперь мы рассмотрим наиболее популярные линейные модели для регрессии.

### Линейная модель регрессии (метод наименьших квадратов)

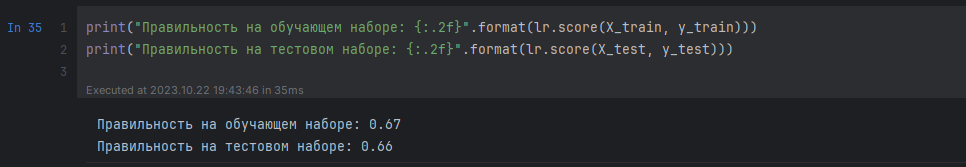
Линейная регрессия или обычный метод наименьших квадратов (ordinary least squares, OLS) – это самый простой и наиболее традиционный метод регрессии. Линейная регрессия находит параметры w и b, которые минимизируют среднеквадратическую ошибку (mean squared error) между спрогнозированными и фактическими ответами у в обучающем наборе. Среднеквадратичная ошибка равна сумме квадратов разностей между спрогнозированными и фактическими значениями. Линейная регрессия проста, что является преимуществом, но в то же время у нее нет инструментов, позволяющих контролировать сложность модели. Ниже приводится программный код, который строит модель, приведенную на рис. 10:



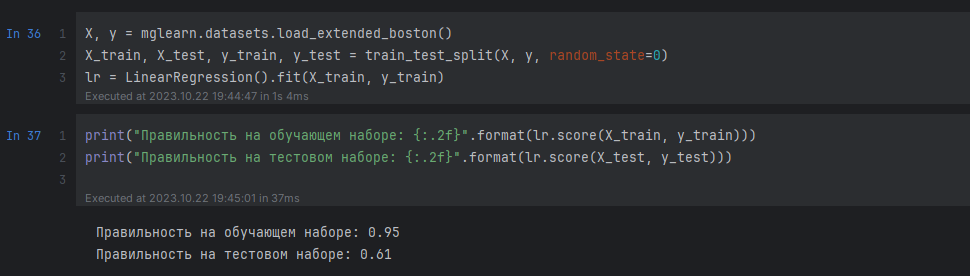
Параметры «наклона» (w), также называемые весами или коэффициентами (coefficients), хранятся в атрибуте coef\_, тогда как сдвиг (offset) или константа (intercept), обозначаемая как b, хранится в атрибуте intercept\_:



Атрибут intercept\_ - это всегда отдельное число с плавающей точкой, тогда как атрибут coef\_ - это массив NumPy, в котором каждому элементу соответствует входной признак. Поскольку в наборе данных wave используется только один входной признак, lr.coef\_ содержит только один элемент. Давайте посмотрим правильность модели на обучающем и тестовом наборах:



Значение R 2 в районе 0.66 указывает на не очень хорошее качество модели, однако можно увидеть, что результаты на обучающем и тестовом наборах очень схожи между собой. Возможно, это указывает на недообучение, а не переобучение. Для этого одномерного массива данных опасность переобучения невелика, поскольку модель очень проста (или строга). Однако для высокоразмерных наборов данных (наборов данных с большим количеством признаков) линейные модели становятся более сложными и существует более высокая вероятность переобучения. Давайте посмотрим, как LinearRegression сработает на более сложном наборе данных, например, на наборе Boston Housing. Вспомним, что этот набор данных имеет 506 примеров (наблюдений) и 105 производных признаков. Во-первых, мы загрузим набор данных и разобьем его на обучающий и тестовый наборы. Затем построим модель линейной регрессии:

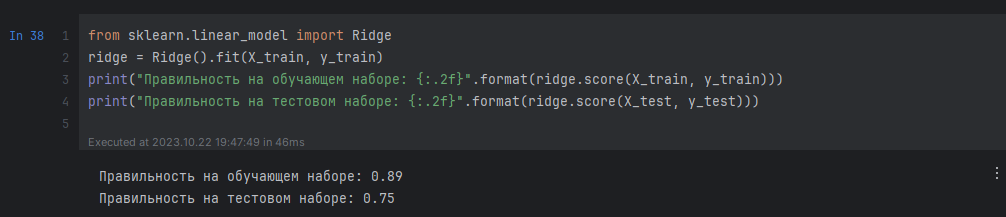


При сравнении правильности на обучающем и тестовом наборах выясняется, что мы очень точно предсказываем на обучающем наборе, однако R2 на тестовом наборе имеет довольно низкое значение.

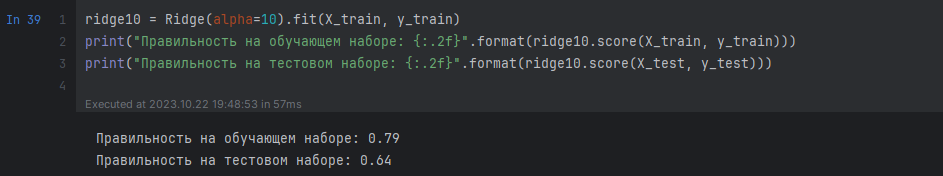
Это несоответствие между правильностью на обучающем наборе и правильностью на тестовом наборе является явным признаком переобучения и поэтому мы должны попытаться найти модель, которая позволит нам контролировать сложность. Одна из наиболее часто используемых альтернатив стандартной линейной регрессии – гребневая регрессия.

### Гребневая регрессия

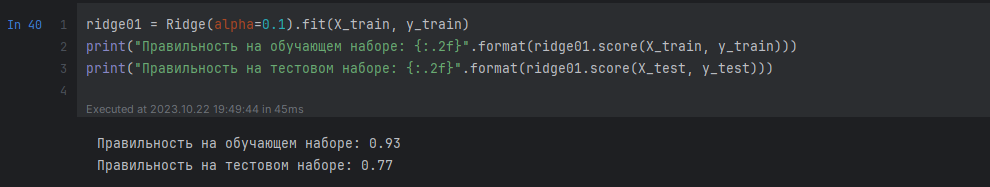
Гребневая регрессия также является линейной моделью регрессии, поэтому ее формула аналогична той, что используется в обычном методе наименьших квадратов. В гребневой регрессии коэффициенты (w) выбираются не только с точки зрения того, насколько хорошо они позволяют предсказывать на обучающих данных, они еще подгоняются в соответствии с дополнительным ограничением. Нам нужно, чтобы величина коэффициентов была как можно меньше. Другими словами, все элементы w должны быть близки к нулю. Это означает, что каждый признак должен иметь как можно меньшее влияние на результат (то есть каждый признак должен иметь небольшой регрессионный коэффициент) и в то же время он должен по-прежнему обладать хорошей прогнозной силой. Это ограничение является примером регуляризации (regularization). Регуляризация означает явное ограничение модели для предотвращения переобучения. Регуляризация, использующаяся в гребневой регрессии, известна как L2 регуляризация. Гребневая регрессии реализована в классе linear\_model.Ridge. Давайте посмотрим, насколько хорошо она работает на расширенном наборе данных **Boston Housing:**



Здесь мы видим, что на обучающем наборе модель Ridge дает меньшую правильность, чем модель LinearRegression, тогда как правильность на тестовом наборе в случае применения гребневой регрессии выше. Это согласуется с нашими ожиданиями. При использовании линейной регрессии мы получили переобучение. Ridge – модель с более строгим ограничением, поэтому меньше вероятность переобучения. Менее сложная модель означает меньшую правильность на обучающем наборе, но лучшую обобщающую способность. Поскольку нас интересует только обобщающая способность, мы должны выбрать модель Ridge вместо модели LinearRegression. Модель Ridge позволяет найти компромисс между простотой модели (получением коэффициентов, близких к нулю) и качеством ее работы на обучающем наборе. Компромисс между простотой модели и качеством работы на обучающем наборе может быть задан пользователем при помощи параметра alpha. В предыдущем примере мы использовали значение параметра по умолчанию alpha=1.0. Впрочем, нет никаких причин считать, что это даст нам оптимальный компромиссный вариант. Оптимальное значение alpha зависит от конкретного используемого набора данных. Увеличение alpha заставляет коэффициенты сжиматься до близких к нулю значений, что снижает качество работы модели на обучающем наборе, но может улучшить ее обобщающую способность. Например:



Уменьшая alpha, мы сжимаем коэффициенты в меньшей степени, что означает движение вправо на рис.1а. При очень малых значениях alpha, ограничение на коэффициенты практически не накладывается, и мы в конечном итоге получаем модель, напоминающую линейную регрессию:



Похоже, что здесь параметр alpha=0.1 сработал хорошо. Мы могли бы попробовать уменьшить alpha еще больше, чтобы улучшить обобщающую способность. Сейчас обратите внимание на то, как параметр alpha соотносится со сложностью модели (рис. 1а). Кроме того, мы можем лучше понять, как параметр alpha меняет модель, использовав атрибут coef\_ с разными значениями alpha. Чем выше alpha, тем более жесткое ограничение накладывается на коэффициенты, поэтому следует ожидать меньшие значения элементов coef\_ для высокого значения alpha. Это подтверждается графиком на рис. 10:

plt.plot(ridge.coef\_, 's', label="Гребневая регрессия alpha=1")  
plt.plot(ridge10.coef\_, '^', label="Гребневая регрессия alpha=10")  
plt.plot(ridge01.coef\_, 'v', label="Гребневая регрессия alpha=0.1")  
plt.plot(lr.coef\_, 'o', label="Линейная регрессия")  
plt.xlabel("Индекс коэффициента")  
plt.ylabel("Оценка коэффициента")  
plt.hlines(0, 0, len(lr.coef\_))  
plt.ylim(-25, 25)  
plt.legend()  
plt.show()

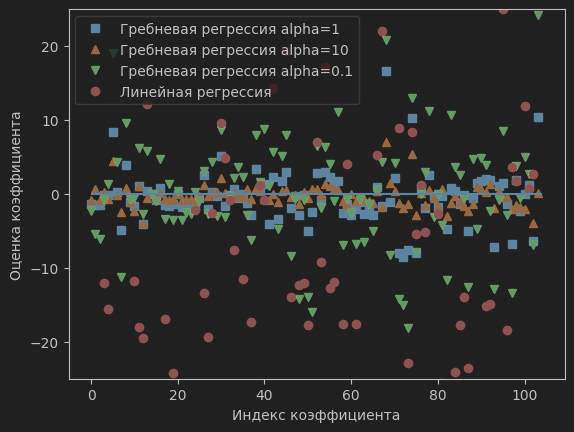
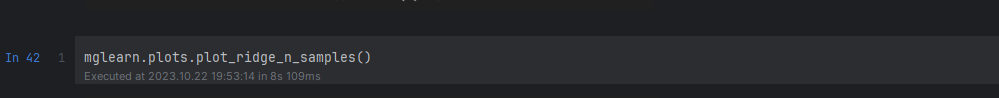
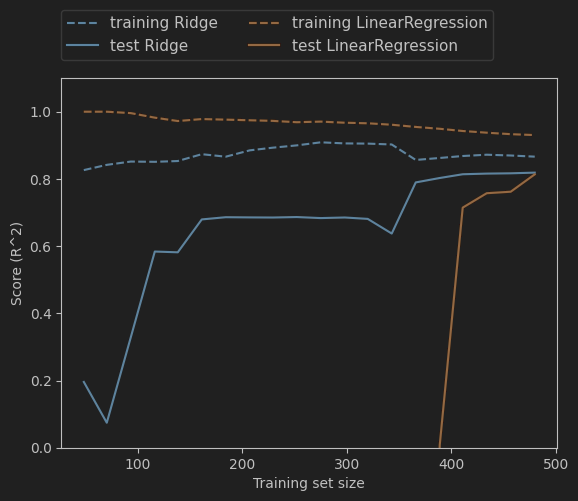


Рис. 10 Сравнение оценок коэффициентов гребневой регрессии с разными значениями alpha и линейной регрессии

Здесь ось х соответствует элементам атрибута coef\_:х=0 показывает коэффициент, связанный с первым признаком, х=1 – коэффициент, связанный со вторым признаком и так далее, вплоть до х=100. Ось y показывает числовые значения соответствующих коэффициентов. Ключевой информацией здесь является то, что для alpha=10 коэффициенты главным образом расположены в диапазоне от -3 до 3. Коэффициенты для модели Ridge с alpha=1 несколько больше. Точки, соответствующие alpha=0.1, имеют более высокие значения, а большинство точек, соответствующих линейной регрессии без регуляризации (что соответствует alpha=0), настолько велики, что находятся за пределами диаграммы. Еще один способ понять влияние регуляризации заключается в том, чтобы зафиксировать значение alpha и при этом менять доступный объем обучающих данных. Мы сформировали выборки разного объема на основе набора данных Boston Housing и затем построили LinearRegression и Ridge(alpha=1) на полученных подмножествах, увеличивая объем. На рис. 11 приводятся графики, которые показывают качество работы модели в виде функции от объема набора данных, их еще называют кривыми обучения (learning curves):



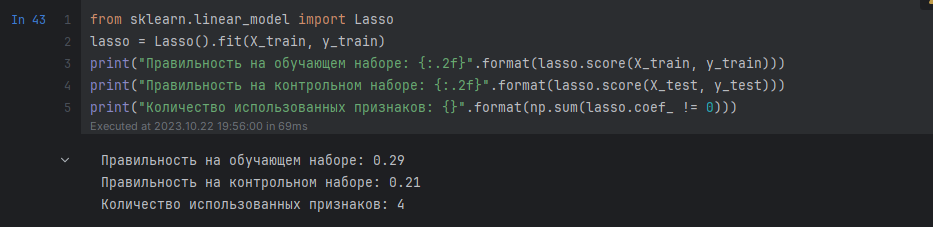


*Рис. 11 Кривые обучения гребневой регрессии и линейной регрессии для набора данных Boston Housing*

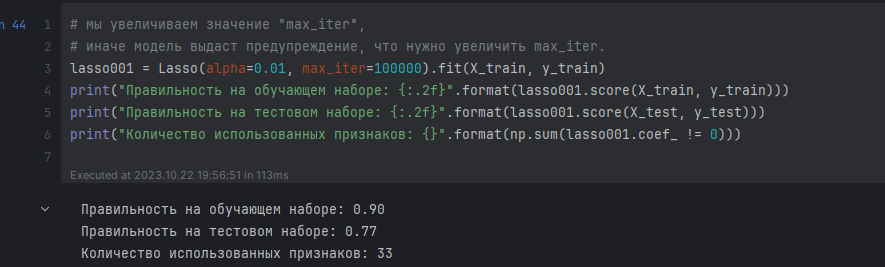
Как и следовало ожидать, независимо от объема данных правильность на обучающем наборе всегда выше правильности на тестовом наборе, как в случае использования гребневой регрессии, так и в случае использования линейной регрессии. Поскольку гребневая регрессия – регуляризированная модель, во всех случаях на обучающем наборе правильность гребневой регрессии ниже правильности линейной регрессии. Однако правильность на тестовом наборе у гребневой регрессии выше, особенно для небольших подмножеств данных. При объеме данных менее 400 наблюдений линейная регрессия не способна обучиться чему-либо. По мере возрастания объема данных, доступного для моделирования, обе модели становятся лучше и в итоге линейная регрессия догоняет гребневую регрессию. Урок здесь состоит в том, что при достаточном объеме обучающих данных регуляризация становится менее важной и при удовлетворительном объеме данных гребневая и линейная регрессии будут демонстрировать одинаковое качество работы (тот факт, что в данном случае это происходит при использовании полного набора данных, является просто случайностью). Еще одна интересная деталь рис. 11 – это снижение правильности линейной регрессии на обучающем наборе. С возрастанием объема данных модели становится все сложнее переобучиться или запомнить данные.

### Лассо

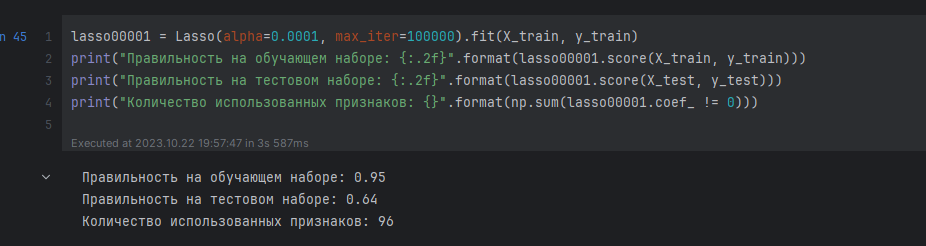
Альтернативой Ridge как метода регуляризации линейной регрессии является Lasso. Как и гребневая регрессия, лассо также сжимает коэффициенты до близких к нулю значений, но несколько иным способом, называемым L1 регуляризацией. Результат L1 регуляризации заключается в том, что при использовании лассо некоторые коэффициенты становятся равны точно нулю. Получается, что некоторые признаки полностью исключаются из модели. Это можно рассматривать как один из видов автоматического отбора признаков. Получение нулевых значений для некоторых коэффициентов часто упрощает интерпретацию модели и может выявить наиболее важные признаки вашей модели. Давайте применим метод лассо к расширенному набору данных **Boston Housing**:



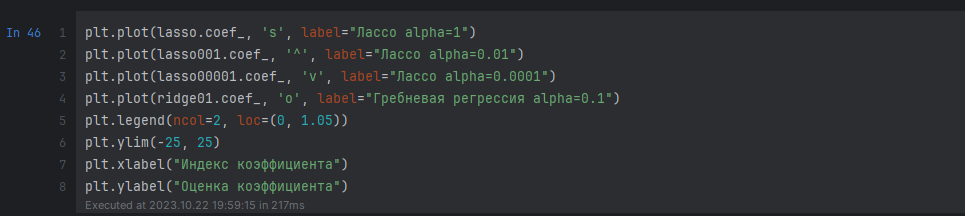
Как видно из сводки, Lasso дает низкую правильность как на обучающем, так и на тестовом наборе. Это указывает на недообучение и мы видим, что из 105 признаков используются только 4. Как и Ridge, Lasso также имеет параметр регуляризации alpha, который определяет степень сжатия коэффициентов до нулевых значений. В предыдущем примере мы использовали значение по умолчанию alpha=1.0. Чтобы снизить недообучение, давайте попробуем уменьшить alpha. При этом нам нужно увеличить значение max\_iter (максимальное количество итераций):

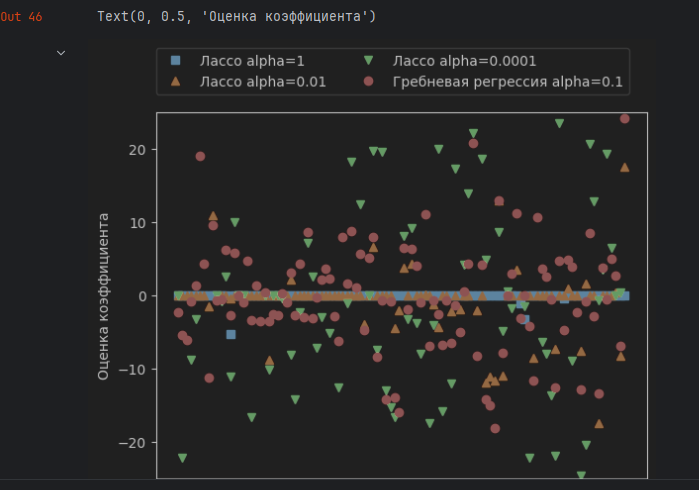


Более низкое значение alpha позволило нам получить более сложную модель, которая продемонстрировала более высокую правильность на обучающем и тестовом наборах. Лассо работает немного лучше, чем гребневая регрессия, и мы используем лишь 33 признака из 105. Это делает данную модель более легкой с точки зрения интерпретации. Однако, если мы установим слишком низкое значение alpha, мы снова нивелируем эффект регуляризации и получим в конечном итоге переобучение, придя к результатам, аналогичным результатам линейной регрессии:



Опять же, мы можем построить графики для коэффициентов различных моделей, аналогичные рис. 10. Результат приведен на рис. 12:





*Рис. 12 Сравнение коэффициентов для лассо-регрессии с разными значениями alpha и гребневой регрессии*

Для alpha=1 мы видим, что не только большинство коэффициентов равны нулю (что мы уже знали), но и остальные коэффициенты также малы по величине. Уменьшив alpha до 0.01, получаем решение, показанное в виде зеленых треугольников, большая часть коэффициентов для признаков становятся в точности равными нулю. При alpha=0.0001 мы получаем практически нерегуляризированную модель, у которой большинство коэффициентов отличны от нуля и имеют большие значения. Для сравнения приводится наилучшее решение, полученное с помощью гребневой регрессии. Модель Ridge с alpha=0.1 имеет такую же прогностическую способность, что и модель лассо с alpha=0.01, однако при использовании гребневой регрессии все коэффициенты отличны от нуля.

На практике, когда стоит выбор между гребневой регрессией и лассо, предпочтение, как правило, отдается гребневой регрессии. Однако, если у вас есть большое количество признаков и есть основания считать, что лишь некоторые из них важны, Lasso может быть оптимальным выбором. Аналогично, если вам нужна легко интерпретируемая модель, Lasso поможет получить такую модель, так как она выберет лишь подмножество входных признаков. В библиотеке scikit-learn также имеется класс ElasticNet, который сочетает в себе штрафы Lasso и Ridge. На практике эта комбинация работает лучше всего, впрочем, это достигается за счет двух корректируемых параметров: один для L1 регуляризации, а другой – для L2 регуляризации.

## Линейные модели для классификации

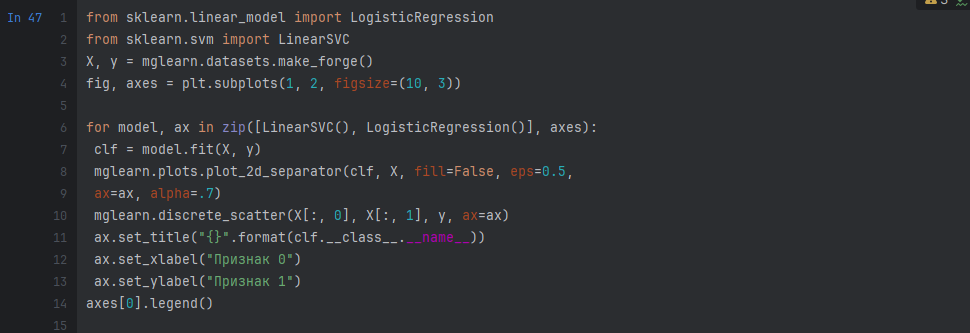
Линейные модели также широко используются в задачах классификации. Давайте посмотрим сначала на бинарную классификацию. В этом случае прогноз получают с помощью следующей формулы:

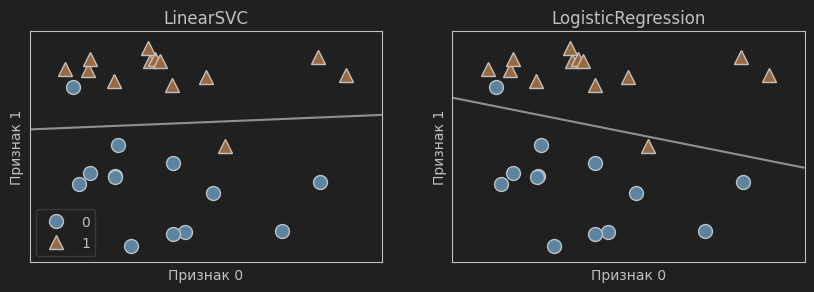
y ˆ = w[0]\* x[0]+ w[1]\* x[1]+...+ w[p]\* x[p]+ b > 0

Формула очень похожа на формулу линейной регрессии, но теперь вместо того, чтобы просто возвратить взвешенную сумму признаков, мы задаем для прогнозируемого значения порог, равный нулю. Если функция меньше нуля, мы прогнозируем класс –1, если она больше нуля, мы прогнозируем класс +1. Это прогнозное правило является общим для всех линейных моделей классификации. Опять же, есть много различных способов найти коэффициенты (w) и константу (b). Для линейных моделей регрессии выход y ˆ является линейной функцией признаков: линией, плоскостью или гиперплоскостью (для большого количества измерений). Для линейных моделей классификации граница принятия решений (decision boundary) является линейной функцией аргумента. Другими словами, (бинарный) линейный классификатор – это классификатор, который разделяет два класса с помощью линии, плоскости или гиперплоскости. В этом разделе мы приведем кокретные примеры. Существует масса алгоритмов обучения линейных моделей. Два критерия задают различия между алгоритмами:

• Измеряемые метрики качества подгонки обучающих данных;

• Факт использования регуляризации и вид регуляризации, если она используется. Различные алгоритмы по-разному определяют, что значит «хорошая подгонка обучающих данных». В силу технико-математических причин невозможно скорректировать w и b, чтобы минимизировать количество неверно классифицированных случаев, выдаваемое алгоритмами, как можно было бы надеяться. С точки зрения поставленных нами целей и различных сфер применения различные варианты метрик качества подгонки (так называемые функции потерь или loss functions) не имеют большого значения. Двумя наиболее распространенными алгоритмами линейной классификации являются логистическая регрессия (logistic regression), реализованная в классе linear\_model. LogisticRegression, и линейный метод опорных векторов (linear support vector machines) или линейный SVM, реализованный в классе svm. LinearSVC (SVC расшифровывается как support vector classifier – классификатор опорных векторов). Несмотря на свое название, логистическая регрессия является алгоритмом классификации, а не алгоритмом регрессии, и его не следует путать с линейной регрессией. Мы можем применить модели LogisticRegression и LinearSVC к набору данных forge и визуализировать границу принятия решений, найденную линейными моделями (рис. 13):



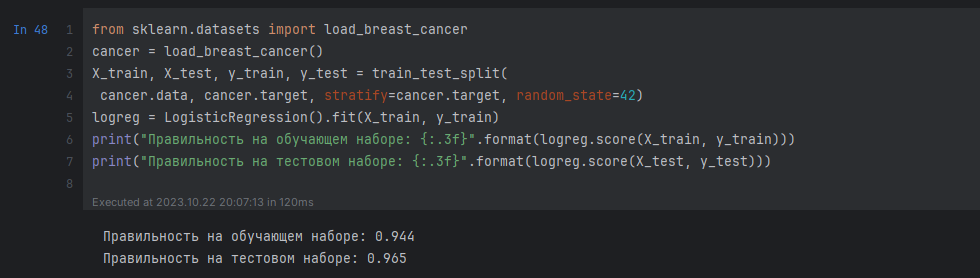


*Рис. 13. Границы принятия решений линейного SVM и логистической регрессии для набора данных forge (использовались значения параметров по умолчанию)*

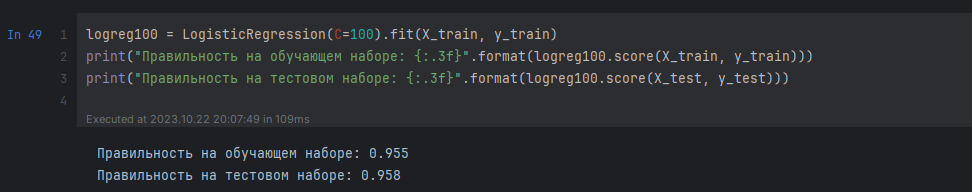
На этом рисунке, как и раньше, первый признак набора данных forge отложен по оси x, а второй признак – по оси y. Здесь показаны границы принятия решений, найденные LinearSVC и LogisticRegression соответственно. Они представлены в виде прямых линий, отделяющих область значений, классифицированных как класс 1 (в верхней части графика) от области значений, классифицированных как класс 0 (в нижней части графика). Другими словами, любая новая точка данных, которая лежит выше черной линии будет отнесена соответствующей моделью к классу 1, тогда как любая точка, лежащая ниже черной линии, будет отнесена к классу 0. Обе модели имеют схожие границы принятия решений. Обратите внимание, что обе модели неправильно классифицировали две точки. По умолчанию обе модели используют L2 регуляризацию, тот же самый метод, который используется в гребневой регрессии.

### Построение модели LogisticRegression

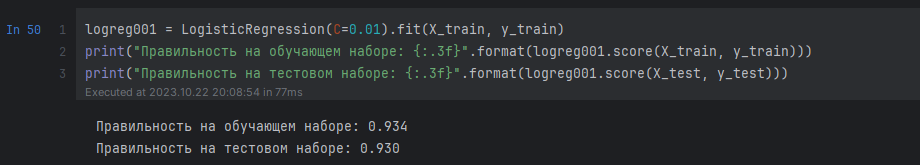
Давайте более подробно разберем работу LogisticRegression на наборе данных Breast Cancer:



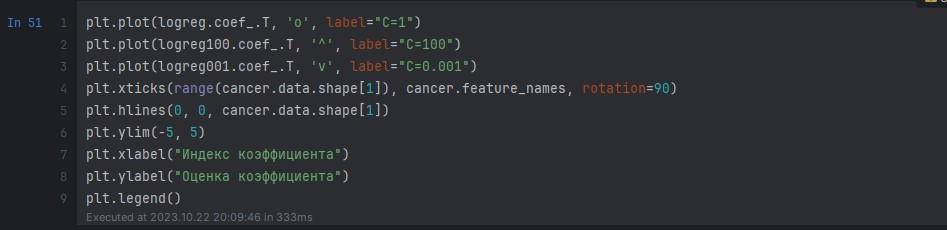
Значение по умолчанию C=1 обеспечивает неплохое качество модели, правильность на обучающем и тестовом наборах составляет 95%. Однако поскольку качество модели на обучающем и тестовом наборах примерно одинаково, вполне вероятно, что мы не дообучили модель. Давайте попробуем увеличить C, чтобы подогнать более гибкую модель:

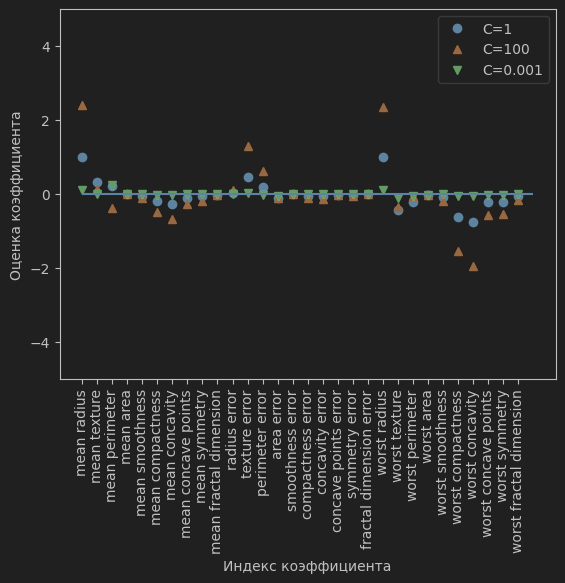


Использование C=100 привело к более высокой правильности на обучающей выборке, а также немного увеличилась правильность на тестовой выборке, что подтверждает наш довод о том, что более сложная модель должна сработать лучше. Кроме того, мы можем выяснить, что произойдет, если мы воспользуемся более регуляризованной моделью (установив C=0.01 вместо значения по умолчанию C=1):



Как и следовало ожидать, когда мы получили недообученную модель и переместились влево по шкале, показанной на рис. 1а, правильность как на обучающем, так и на тестовом наборах снизилась по сравнению с правильностью, которую мы получили, использовав параметры по умолчанию. И, наконец, давайте посмотрим на коэффициенты логистической регрессии, полученные с использованием трех различных значений параметра регуляризации C (рис. 14):





*Рис. 14 Коэффициенты, полученные с помощью логистической регрессии с разными значениями C для набора данных Breast Cancer*

# Задание:

1. Изучите материал, представленный в разделе Ход работы[[1]](#footnote-1).

|  |  |
| --- | --- |
| [**load\_diabetes**](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes)(\*[, return\_X\_y, as\_frame]) | Загрузите и верните набор данных диабета (регрессия). |
| [**load\_linnerud**](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_linnerud.html#sklearn.datasets.load_linnerud)(\*[, return\_X\_y, as\_frame]) | Загрузите и верните набор данных linnerud физических упражнений. |

1. Подключите датасеты моделей регрессии:
2. Изучите подключенные данные.
3. Постройте модели регрессии (**KNeighborsRegressor, LinearRegression, Ridge, Lasso, LogisticRegression**) для полученных датасетов, описанные в Ходе работы.
   1. Настройте наилучшие параметры моделей.
   2. Сравните результаты и сделайте выводы.
4. Оформите Notebook.
5. Оформите выводы по работе.

20 баллов -лаб 4

25 баллов - лаб 5

1. Источник: Andreas C. Mueller and Sarah Guido. Introduction to Machine Learning with Python. O’Reilly 2016 [↑](#footnote-ref-1)