Лабораторная работа №4

Линейные модели. Метод наименьших квадратов

Линейные модели представляют собой класс моделей, которые широко используются на практике и были предметом детального изучения в течение последних нескольких десятилетий, а их история насчитывает более ста лет. Линейные модели дают прогноз, используя ***линейную функцию*** *(linear function)* входных признаков, о которой мы расскажем ниже.

Линейные модели для регрессии

Для регрессии общая прогнозная формула линейной модели выглядит следующим образом:

$$\hat{y}\ w[0] * x[0]\ \ w[1] * x[1]\ \ \ldots.\ w[p] * x[p]\ b$$

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

$$\hat{y}\ w[0] * x[0]\ \ b$$

Возможно из школьного курса математики вы вспомните, что эта формула – уравнение прямой. Здесь ***x[0]*** является наклоном, а ***b*** – сдвигом по оси y. 9 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. 4.1):

```
mglearn.plots.plot_linear_regression_wave()
```

```
w[0]: 0.393906 b: -0.031804
```

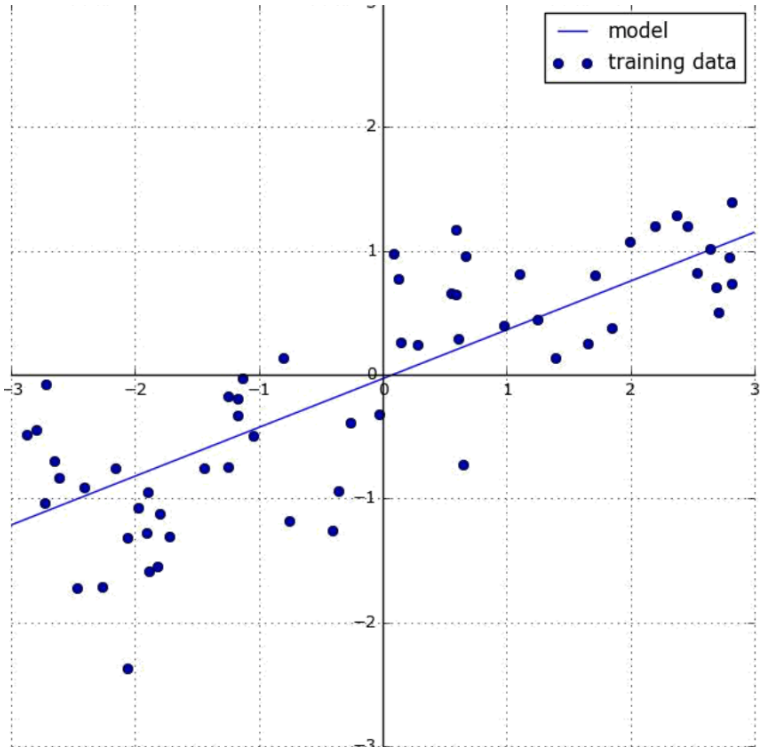

Рис. 4.1 Прогнозы линейной модели для набора данных wave

Мы добавили координатный крест на график, чтобы прямую было проще интерпретировать. Взглянув на значение w[0], мы видим, что наклон должен быть около 0.4 и это визуально подтверждается на графике. Константа (место пересечения линии прогнозов с осью ординат) чуть меньше нуля, что также подтверждается графиком.

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, 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 y 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 y можно прекрасно смоделировать (на обучающей выборке) в виде линейной функции.10

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 регрессия

(обычный 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 y 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. 4.1:

```
from sklearn.linear_model import LinearRegression
X, y = mglearn.datasets.make_wave(n_samples=60)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
lr = LinearRegression().fit(X_train, y_train)
```

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_**:

```
print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))
```

```
lr.coef_: [ 0.394]
lr.intercept_: -0.031804343026759746
```

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:

```
print("Правильность на обучающем наборе: {:.2f}".format(lr.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(lr.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.67
Правильность на тестовом наборе: 0.66
```

Значение R2 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 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:

```
X, y = mglearn.datasets.load_extended_boston()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lr = LinearRegression().fit(X_train, y_train)
```

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:

```
print("Правильность на обучающем наборе: {:.2f}".format(lr.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(lr.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.95
Правильность на тестовом наборе: 0.61
```

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, которую мы рассмотрим ниже.

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) 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:

```
from sklearn.linear_model import Ridge
ridge = Ridge().fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(ridge.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(ridge.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.89
Правильность на тестовом наборе: 0.75
```

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:

```
ridge10 = Ridge(alpha=10).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(ridge10.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(ridge10.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.79
Правильность на тестовом наборе: 0.64
```

Уменьшая alpha, мы сжимаем коэффициенты в меньшей степени, что означает движение вправо на рис. 2.1. При очень малых значениях alpha, ограничение на коэффициенты практически не накладывается и мы в конечном итоге получаем модель, напоминающую линейную регрессию:

```
ridge10 = Ridge(alpha=0.1).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(ridge10.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(ridge10.score(X_test, y_test)))
```

```
Правильность на обучающем наборе: 0.93
Правильность на тестовом наборе: 0.77
```

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.

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. 4.2:

```
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()
```

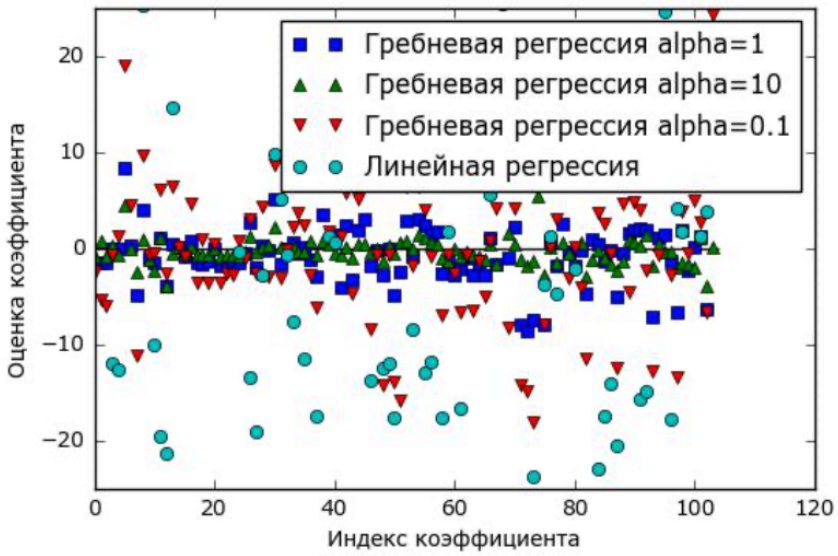

Рис. 4.2 Сравнение оценок коэффициентов гребневой регрессии с разными значениями alpha и линейной регрессии

Здесь ось x соответствует элементам атрибута coef_:x=0 показывает коэффициент, связанный с первым признаком, x=1 – коэффициент, связанный со вторым признаком и так далее, вплоть до x=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. 2.13 приводятся графики, которые показывают качество работы модели в виде функции от объема набора данных, их еще называют ***кривыми обучения*** *(learning curves)*:

```
mglearn.plots.plot_ridge_n_samples()
```

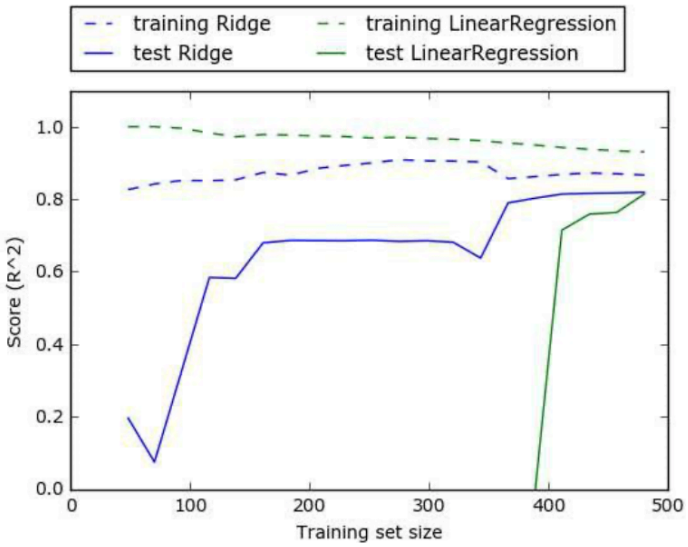

Рис. 4.3 Кривые обучения гребневой регрессии и линейной регрессии для набора данных Boston Housing

Как и следовало ожидать, независимо от объема данных правильность на обучающем наборе всегда выше правильности на тестовом наборе, как 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. 2.13 – 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:

```
from sklearn.linear_model import Lasso

lasso = Lasso().fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(lasso.score(X_train, y_train)))
print("Правильность на контрольном наборе: {:.2f}".format(lasso.score(X_test, y_test)))
print("Количество использованных признаков: {}".format(np.sum(lasso.coef_ != 0)))
```

```
Правильность на обучающем наборе: 0.29
Правильность на контрольном наборе: 0.21
Количество использованных признаков: 4
```

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):

```
# мы увеличиваем значение "max_iter",
# иначе модель выдаст предупреждение, что нужно увеличить max_iter.
lasso001 = Lasso(alpha=0.01, max_iter=100000).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(lasso001.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(lasso001.score(X_test, y_test)))
print("Количество использованных признаков: {}".format(np.sum(lasso001.coef_ != 0)))
```

```
Правильность на обучающем наборе: 0.90
Правильность на контрольном наборе: 0.77
Количество использованных признаков: 33
```

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:

```
lasso00001 = Lasso(alpha=0.0001, max_iter=100000).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(lasso00001.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(lasso00001.score(X_test, y_test)))
print("Количество использованных признаков: {}".format(np.sum(lasso00001.coef_ != 0)))
```

```
Правильность на обучающем наборе: 0.95
Правильность на контрольном наборе: 0.64
Количество использованных признаков: 94
```

Опять же, мы можем построить графики для коэффициентов различных моделей, аналогичные рис. 4.2. Результат приведен на рис. 4.4:

```
plt.plot(lasso.coef_, 's', label="Лассо alpha=1")
plt.plot(lasso001.coef_, '^', label="Лассо alpha=0.01")
plt.plot(lasso00001.coef_, 'v', label="Лассо alpha=0.0001")

plt.plot(ridge01.coef_, 'o', label="Гребневая регрессия alpha=0.1")
plt.legend(ncol=2, loc=(0, 1.05))
plt.ylim(-25, 25)
plt.xlabel("Индекс коэффициента")
plt.ylabel("Оценка коэффициента")
```

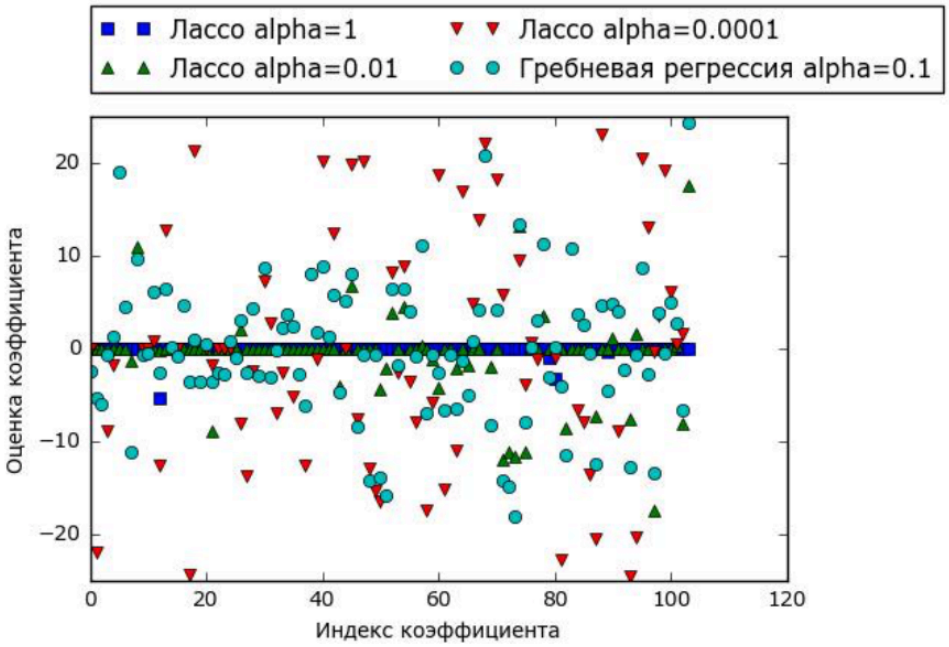

Рис. 4.4 Сравнение коэффициентов для лассо-регрессии с разными значениями alpha и гребневой регрессии

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

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.

Код к лабораторной работе:

```
import sklearn
import mglearn
import matplotlib.pyplot as plt
import matplotlib
import numpy as np

mglearn.plots.plot_linear_regression_wave()
plt.show()

from sklearn.linear_model import LinearRegression
X, y = mglearn.datasets.make_wave(n_samples=60)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

lr = LinearRegression().fit(X_train, y_train)

print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))

print("Правильность на обучающем наборе: {:.2f}".format(lr.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(lr.score(X_test, y_test)))

X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lr = LinearRegression().fit(X_train, y_train)

print("Правильность на обучающем наборе: {:.2f}".format(lr.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(lr.score(X_test, y_test)))

from sklearn.linear_model import Ridge
ridge = Ridge().fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(ridge.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(ridge.score(X_test, y_test)))

ridge10 = Ridge(alpha=10).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(ridge10.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(ridge10.score(X_test, y_test)))

ridge01 = Ridge(alpha=0.1).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(ridge01.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(ridge01.score(X_test, y_test)))

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()

mglearn.plots.plot_ridge_n_samples()
plt.show()

from sklearn.linear_model import Lasso
lasso = Lasso().fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(lasso.score(X_train, y_train)))
print("Правильность на контрольном наборе: {:.2f}".format(lasso.score(X_test, y_test)))
print("Количество использованных признаков: {}".format(np.sum(lasso.coef_ != 0)))

lasso001 = Lasso(alpha=0.01, max_iter=100000).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(lasso001.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(lasso001.score(X_test, y_test)))
print("Количество использованных признаков: {}".format(np.sum(lasso001.coef_ != 0)))

lasso00001 = Lasso(alpha=0.0001, max_iter=100000).fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(lasso00001.score(X_train, y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(lasso00001.score(X_test, y_test)))
print("Количество использованных признаков: {}".format(np.sum(lasso00001.coef_ != 0)))

plt.plot(lasso.coef_, 's', label="Лассо alpha=1")
plt.plot(lasso001.coef_, '^', label="Лассо alpha=0.01")
plt.plot(lasso00001.coef_, 'v', label="Лассо alpha=0.0001")

plt.plot(ridge01.coef_, 'o', label="Гребневая регрессия alpha=0.1")
plt.legend(ncol=2, loc=(0, 1.05))
plt.ylim(-25, 25)
plt.xlabel("Индекс коэффициента")
plt.ylabel("Оценка коэффициента")
plt.show()
```