**Линейные модели** являются одним из фундаментальных инструментов машинного обучения, особенно для задач **регрессии**, где целью является предсказание *непрерывного числового значения*. Основная идея заключается в том, чтобы **смоделировать выходное значение как линейную комбинацию входных признаков**. Для одного признака x[0] модель выглядит как простое уравнение прямой: y^ = w[0] \* x[0] + b. При наличии нескольких признаков (x[0] до x[p]) формула обобщается: y^ = w[0]\*x[0] + ... + w[p]\*x[p] + b. Здесь w (веса или коэффициенты) отражают важность или влияние каждого признака на прогноз, а b (свободный член или сдвиг) – это базовое значение прогноза, когда все признаки равны нулю. Обучение такой модели сводится к нахождению **оптимальных значений w и b** на основе имеющихся данных.

Самый базовый и классический подход к нахождению этих параметров – **Обычный метод наименьших квадратов (OLS),** реализованный в **LinearRegression**. Его цель – минимизировать среднеквадратичную ошибку (MSE), то есть сумму квадратов разностей между реальными значениями (y) и предсказанными моделью (y^) на обучающем наборе данных. Это интуитивно понятный метод, но у него есть существенный недостаток: **он не контролирует сложность модели**. На наборах данных с большим количеством признаков (особенно если их число сравнимо или превышает количество примеров) **OLS склонен к переобучению** (overfitting). Модель слишком точно подстраивается под обучающие данные, "запоминая" их шум и особенности, и в результате плохо работает на новых, ранее не виденных данных. Это проявляется в высокой точности на обучающей выборке и значительно более низкой – на тестовой.

Чтобы бороться с переобучением и улучшить обобщающую способность модели (ее умение хорошо работать на новых данных), применяют **регуляризацию**. Идея регуляризации состоит во **внесении изменений в процесс обучения, которые ограничивают сложность модели**. Это достигается путем добавления к функции ошибки (MSE) **специального штрафа**, **величина которого зависит от значений коэффициентов w**. Теперь модель при обучении стремится не только минимизировать ошибку предсказания, но и удерживать коэффициенты w небольшими по величине. Степень этого ограничения контролируется **параметром alpha**: чем больше alpha, тем сильнее штраф и тем проще (более ограниченной) становится модель. **Однако слишком большое значение alpha может привести к недообучению** (underfitting), когда модель становится чрезмерно простой и плохо описывает даже обучающие данные.

Существуют два основных типа регуляризации для линейных моделей:

1. **Гребневая регрессия** (Ridge Regression): Использует L2-регуляризацию, где **штраф пропорционален *сумме квадратов* коэффициентов** (alpha \* sum(w²)). Ridge эффективно "ужимает" **коэффициенты, делая их меньше** по абсолютной величине. Это помогает стабилизировать модель и снизить ее сложность, но коэффициенты редко становятся *в точности* равными нулю. Ridge часто является хорошим выбором по умолчанию для улучшения обобщающей способности линейных моделей.
2. **Лассо регрессия** (Lasso Regression): Использует L1-регуляризацию, где штраф пропорционален *сумме абсолютных значений* коэффициентов (alpha \* sum(|w|)). Ключевое отличие Lasso от Ridge заключается в том, что L1-штраф **способен "обнулять" некоторые коэффициенты полностью**. Таким образом, Lasso не только уменьшает сложность модели, но и **выполняет автоматический отбор признаков**, оставляя в модели только те, которые вносят наиболее существенный вклад в прогноз. Это делает модель потенциально более простой для интерпретации. Lasso особенно полезен, когда имеется большое количество признаков, и есть основания полагать, что лишь немногие из них действительно важны.

В практическом плане, выбор между Ridge и Lasso зависит от задачи. Если важно сохранить все признаки, но уменьшить их влияние, Ridge может быть предпочтительнее. Если же целью является получение более простой, разреженной модели с отбором наиболее значимых признаков, то стоит попробовать Lasso. В обоих случаях подбор оптимального значения alpha является важным шагом для достижения наилучшего баланса между точностью и обобщающей способностью.

Хорошо, давайте пройдем по коду и графикам ОЧЕНЬ подробно, шаг за шагом.

**Шаг 1: Простая Линейная Регрессия (данные wave)**

# Импортируем нужные инструменты

import sklearn

import mglearn # Библиотека для визуализации из книги

import matplotlib.pyplot as plt # Для рисования графиков

import matplotlib

import numpy as np # Для математических операций

# --- ГРАФИК 1 ---

# Эта команда из mglearn сама строит график для данных 'wave'

# Она показывает точки данных и линию регрессии OLS (Обычный Метод Наименьших Квадратов)

mglearn.plots.plot\_linear\_regression\_wave()

plt.show() # Показать график

# --- ПОДГОТОВКА ДАННЫХ И ОБУЧЕНИЕ ---

from sklearn.linear\_model import LinearRegression # Импортируем саму модель OLS

# Загружаем простой набор данных 'wave'. У него 1 входной признак (X) и 1 выход (y). 60 точек.

X, y = mglearn.datasets.make\_wave(n\_samples=60)

from sklearn.model\_selection import train\_test\_split # Инструмент для разделения данных

# Делим данные на две части: обучающую (на ней модель учится) и тестовую (на ней проверяем)

# random\_state=42 гарантирует, что деление будет одинаковым при каждом запуске

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42)

# Создаем модель LinearRegression и сразу обучаем ее на обучающих данных (X\_train, y\_train)

# .fit() - это команда "учиться"

lr = LinearRegression().fit(X\_train, y\_train)

# --- РЕЗУЛЬТАТЫ МОДЕЛИ OLS ---

# Выводим параметры, которые модель нашла:

# coef\_ это 'w' (коэффициент наклона), т.к. у нас 1 признак, это будет одно число

print("lr.coef\_: {}".format(lr.coef\_))

# intercept\_ это 'b' (сдвиг по оси Y, где линия пересекает вертикальную ось)

print("lr.intercept\_: {}".format(lr.intercept\_))

# Выводим оценку качества модели (R^2 score). Ближе к 1 - лучше.

# Сначала на обучающем наборе (на котором учились)

print("Правильность на обучающем наборе: {:.2f}".format(lr.score(X\_train, y\_train)))

# Потом на тестовом наборе (новые данные)

print("Правильность на тестовом наборе: {:.2f}".format(lr.score(X\_test, y\_test)))

**Объяснение Графика 1 (plot\_linear\_regression\_wave) и Вывода 1:**

* **Что на графике?** Вы видите синие точки и синюю прямую линию.
* **Синие точки:** Это данные из make\_wave. Горизонтальная позиция точки - это значение X, вертикальная - значение y. Точки образуют волну.
* **Синяя линия:** Это предсказание модели LinearRegression. Она может предсказывать *только* в виде прямой линии. Эта линия проведена так, чтобы быть как можно ближе ко всем синим точкам в среднем (по методу наименьших квадратов).
* **Связь с выводом:**
  + lr.coef\_: [0.394] - Это **наклон** синей линии на графике. Положительный наклон означает, что линия идет вверх при движении вправо.
  + lr.intercept\_: -0.0318 - Это точка, где синяя линия **пересекает вертикальную ось Y** (при X=0). Она чуть ниже нуля.
  + Правильность на обучающем наборе: 0.67
  + Правильность на тестовом наборе: 0.66  
    Эти числа (R² score) показывают, насколько хорошо линия описывает точки. Значение около 0.66 не очень высокое. **График объясняет почему:** прямая линия - слишком грубое приближение для волнистых данных. Она не может повторить изгибы.
* **Главный вывод из этого шага:** Простая линейная регрессия (OLS) находит наилучшую прямую линию, но если данные нелинейные (как волна), то прямая линия будет плохой моделью (**недообучение**). Оценки на обучении и тесте близки, значит, **переобучения здесь нет**.

**Шаг 2: Линейная Регрессия на Сложных Данных (Boston Housing)**

# --- ПЕРЕКЛЮЧАЕМСЯ НА ДРУГОЙ ДАТАСЕТ ---

# Загружаем 'расширенный' Бостонский датасет. В нем МНОГО признаков (105 штук).

X, y = mglearn.datasets.load\_extended\_boston()

# Снова делим на обучение и тест (random\_state=0 для воспроизводимости этого шага)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

# Снова обучаем ПРОСТУЮ LinearRegression (OLS) на этих СЛОЖНЫХ данных

lr = LinearRegression().fit(X\_train, y\_train)

# --- РЕЗУЛЬТАТЫ OLS НА СЛОЖНЫХ ДАННЫХ ---

# Оцениваем качество

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)))

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

IGNORE\_WHEN\_COPYING\_END

**Объяснение Вывода 2 (Графика здесь нет):**

* **Что делает код:** Мы взяли тот же простой метод LinearRegression (OLS), но применили его к сложному датасету Boston Housing со 105 признаками.
* **Результаты в выводе:**
  + Правильность на обучающем наборе: 0.95 - **Очень высокое** значение! Модель почти идеально подогналась под данные, на которых училась.
  + Правильность на тестовом наборе: 0.61 - **Значительно ниже!** На новых данных модель работает гораздо хуже.
* **Главный вывод из этого шага:** Когда признаков много, простая LinearRegression может **переобучиться**. Она становится слишком сложной, "запоминает" обучающие данные вместо того, чтобы найти общие закономерности. **Большой разрыв** между качеством на обучении (0.95) и тесте (0.61) - это явный **признак переобучения**.

**Шаг 3: Гребневая Регрессия (Ridge) для борьбы с переобучением**

# --- ИСПОЛЬЗУЕМ RIDGE РЕГРЕССИЮ ---

from sklearn.linear\_model import Ridge # Импортируем Ridge

# Обучаем Ridge с alpha=1.0 (значение по умолчанию)

ridge = Ridge().fit(X\_train, y\_train)

# Оцениваем качество Ridge(alpha=1)

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)))

# Обучаем Ridge с alpha=10 (БОЛЬШЕ alpha = СИЛЬНЕЕ регуляризация)

ridge10 = Ridge(alpha=10).fit(X\_train, y\_train)

# Оцениваем качество Ridge(alpha=10)

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)))

# Обучаем Ridge с alpha=0.1 (МЕНЬШЕ alpha = СЛАБЕЕ регуляризация)

ridge01 = Ridge(alpha=0.1).fit(X\_train, y\_train)

# Оцениваем качество Ridge(alpha=0.1)

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)))

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

IGNORE\_WHEN\_COPYING\_END

**Объяснение Вывода 3, 4, 5 (Графика пока нет):**

* **Что делает код:** Мы используем Ridge регрессию. Она похожа на LinearRegression, но добавляет "штраф" за слишком большие коэффициенты w, чтобы модель не переобучалась. Параметр alpha контролирует силу этого штрафа.
* **Результаты в выводе:**
  + **Ridge (alpha=1):** Обучение: 0.89, Тест: 0.75. Сравните с OLS (0.95, 0.61). Обучающая оценка стала *ниже* (это нормально, регуляризация мешает идеальной подгонке), но тестовая оценка стала *выше*! Разрыв между ними уменьшился. **Ridge помог!**
  + **Ridge (alpha=10):** Обучение: 0.79, Тест: 0.64. Сильный штраф. Обучающая оценка еще ниже. Тестовая тоже упала по сравнению с alpha=1. Возможно, штраф **слишком сильный**, модель стала чрезмерно простой (недообучается).
  + **Ridge (alpha=0.1):** Обучение: 0.93, Тест: 0.77. Слабый штраф. Обучающая оценка ближе к OLS. Тестовая оценка **самая лучшая** из всех Ridge! Похоже, alpha=0.1 - хороший компромисс.
* **Главный вывод из этого шага:** Ridge регрессия с правильно подобранным alpha (здесь alpha=0.1) может **уменьшить переобучение** (сократить разрыв между оценками на обучении и тесте) и **улучшить качество** на новых данных по сравнению с OLS.

**Шаг 4: Визуализация Коэффициентов OLS и Ridge**

# --- ГРАФИК 2 ---

# Рисуем значения коэффициентов 'w' для разных моделей

plt.plot(ridge.coef\_, 's', label="Гребневая регрессия alpha=1") # Квадраты для alpha=1

plt.plot(ridge10.coef\_, '^', label="Гребневая регрессия alpha=10") # Треугольники вверх для alpha=10

plt.plot(ridge01.coef\_, 'v', label="Гребневая регрессия alpha=0.1") # Треугольники вниз для alpha=0.1

plt.plot(lr.coef\_, 'o', label="Линейная регрессия") # Круги для OLS (LinearRegression)

plt.xlabel("Индекс коэффициента") # Подпись оси X

plt.ylabel("Оценка коэффициента") # Подпись оси Y

plt.hlines(0, 0, len(lr.coef\_)) # Рисуем горизонтальную линию на уровне нуля

plt.ylim(-25, 25) # Ограничиваем вертикальную ось, чтобы было лучше видно

plt.legend() # Показать легенду (какая форма что означает)

plt.show() # Показать график

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

IGNORE\_WHEN\_COPYING\_END

**Объяснение Графика 2:**

* **Что на графике?** Много точек разной формы и цвета + горизонтальная линия.
* **Оси:**
  + Горизонтальная ось (X): Номер признака (от 0 до 104). Каждый номер - это один из 105 признаков датасета Boston.
  + Вертикальная ось (Y): Значение коэффициента w, которое модель присвоила этому признаку. Чем дальше точка от нуля (вверх или вниз), тем "важнее" модель считает этот признак.
* **Точки:**
  + **Синие круги (OLS):** Коэффициенты модели LinearRegression, которая переобучилась. Видите, как они разбросаны? Многие очень большие (далеко от нуля). Это значит, OLS придал слишком большое значение многим признакам.
  + **Красные квадраты (Ridge alpha=1):** Коэффициенты модели Ridge с alpha=1. Они явно **ближе к нулю**, чем синие круги. Ridge "сжал" их.
  + **Зеленые треугольники вверх (Ridge alpha=10):** Сильная регуляризация. Точки **очень сильно прижаты к нулю**. Модель считает почти все признаки малозначимыми.
  + **Пурпурные треугольники вниз (Ridge alpha=0.1):** Слабая регуляризация. Точки ближе к синим кругам OLS, чем красные или зеленые, но все же в целом меньше по модулю.
* **Горизонтальная линия Y=0:** Показывает нулевое значение коэффициента (признак не влияет на результат).
* **Главный вывод из этого графика:** Он **визуально показывает эффект регуляризации Ridge**: она уменьшает абсолютные значения коэффициентов (w), "прижимая" их к нулю. Чем больше alpha, тем сильнее это "сжатие". Это делает модель менее чувствительной к отдельным признакам и помогает бороться с переобучением.

**Шаг 5: Кривые Обучения (Сравнение OLS и Ridge)**

# --- ГРАФИК 3 ---

# Используем готовую функцию из mglearn для построения кривых обучения

# Она обучает OLS и Ridge(alpha=1) на разных по размеру частях обучающих данных

# и строит графики качества на обучении и тесте для каждого размера

mglearn.plots.plot\_ridge\_n\_samples()

plt.show() # Показать график

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

IGNORE\_WHEN\_COPYING\_END

**Объяснение Графика 3:**

* **Что на графике?** Четыре линии (2 пунктирные, 2 сплошные).
* **Оси:**
  + Горизонтальная ось (X): Размер обучающей выборки (сколько данных использовалось для обучения на данном шаге).
  + Вертикальная ось (Y): Качество модели (R² score).
* **Линии:**
  + **Пунктирные:** Качество на **обучающих** данных.
  + **Сплошные:** Качество на **тестовых** данных (самое важное!).
  + **Синие (training/test LinearRegression):** Это для OLS.
    - *Пунктирная синяя:* Начинается очень высоко (модель легко запоминает мало данных) и падает.
    - *Сплошная синяя:* Начинается низко, растет, но всегда **существенно ниже** пунктирной синей. **Этот большой разрыв - переобучение.**
  + **Зеленые (training/test Ridge):** Это для Ridge (alpha=1).
    - *Пунктирная зеленая:* Начинается ниже синей (Ridge мешает запоминанию).
    - *Сплошная зеленая:* Растет и большую часть времени идет **выше** сплошной синей. **Разрыв** между пунктирной и сплошной зеленой **гораздо меньше**.
* **Главный вывод из этого графика:** Ridge (зеленые линии) показывает **лучшее качество на тестовых данных** (выше сплошная линия) и **меньше переобучается** (меньше разрыв между линиями), чем OLS (синие линии), особенно когда обучающих данных не очень много.

**Шаг 6: Лассо Регрессия (Lasso)**

# --- ИСПОЛЬЗУЕМ LASSO РЕГРЕССИЮ ---

from sklearn.linear\_model import Lasso # Импортируем Lasso

# Обучаем Lasso с alpha=1.0 (по умолчанию)

lasso = Lasso().fit(X\_train, y\_train)

# Оцениваем качество Lasso(alpha=1)

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)))

# !!! Считаем, сколько коэффициентов НЕ РАВНЫ НУЛЮ !!!

print("Количество использованных признаков: {}".format(np.sum(lasso.coef\_ != 0)))

# Обучаем Lasso с alpha=0.01 (слабее регуляризация)

# Lasso иногда требует больше итераций для сходимости при малых alpha

lasso001 = Lasso(alpha=0.01, max\_iter=100000).fit(X\_train, y\_train)

# Оцениваем качество Lasso(alpha=0.01)

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)))

# Обучаем Lasso с alpha=0.0001 (очень слабая регуляризация)

lasso00001 = Lasso(alpha=0.0001, max\_iter=100000).fit(X\_train, y\_train)

# Оцениваем качество Lasso(alpha=0.0001)

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)))

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

IGNORE\_WHEN\_COPYING\_END

**Объяснение Вывода 6, 7, 8 (Графика пока нет):**

* **Что делает код:** Мы используем Lasso регрессию. Это другой тип регуляризации (L1). Ее особенность в том, что она может делать коэффициенты w **в точности равными нулю**.
* **Результаты в выводе:**
  + **Lasso (alpha=1):** Обучение: 0.29, Тест: 0.21. Качество **очень низкое**. Признаков использовано: 4. Это значит, что из 105 признаков Lasso оставил только 4, а коэффициенты остальных 101 сделал **равными нулю**. Регуляризация слишком сильная.
  + **Lasso (alpha=0.01):** Обучение: 0.90, Тест: 0.77. Качество **хорошее**, сравнимо с лучшим Ridge. Признаков использовано: 33. Lasso все еще выполнил **отбор признаков**, но оставил больше, чем при alpha=1.
  + **Lasso (alpha=0.0001):** Обучение: 0.95, Тест: 0.64. Качество **похоже на OLS** (переобучение). Признаков использовано: 94. Почти все признаки оставлены, регуляризация почти не сработала.
* **Главный вывод из этого шага:** Lasso не только регулирует модель, но и выполняет **автоматический отбор признаков**, обнуляя коэффициенты для "ненужных" (по его мнению) признаков. Сила отбора зависит от alpha.

**Шаг 7: Визуализация Коэффициентов Lasso и Ridge**

# --- ГРАФИК 4 ---

# Сравниваем коэффициенты Lasso и лучшей модели Ridge

plt.plot(lasso.coef\_, 's', label="Лассо alpha=1") # Квадраты для Lasso(alpha=1)

plt.plot(lasso001.coef\_, '^', label="Лассо alpha=0.01") # Треугольники вверх для Lasso(alpha=0.01)

plt.plot(lasso00001.coef\_, 'v', label="Лассо alpha=0.0001") # Треугольники вниз для Lasso(alpha=0.0001)

plt.plot(ridge01.coef\_, 'o', label="Гребневая регрессия alpha=0.1") # Круги для лучшей Ridge

plt.legend(ncol=2, loc=(0, 1.05)) # Легенда в 2 колонки сверху

plt.ylim(-25, 25)

plt.xlabel("Индекс коэффициента")

plt.ylabel("Оценка коэффициента")

plt.show() # Показать график

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

IGNORE\_WHEN\_COPYING\_END

**Объяснение Графика 4:**

* **Что на графике?** Снова график коэффициентов, но теперь сравниваем Lasso и Ridge.
* **Оси:** Те же: X - номер признака, Y - значение коэффициента.
* **Точки:**
  + **Синие круги (Ridge alpha=0.1):** Наша лучшая Ridge модель. Коэффициенты маленькие, но почти все **не равны нулю** (точки не лежат точно на горизонтальной линии Y=0).
  + **Красные квадраты (Lasso alpha=1):** Посмотрите внимательно: подавляющее большинство квадратов лежат **точно на линии Y=0**. Только 4 точки (как мы видели в выводе) имеют ненулевые коэффициенты. Lasso "выбросил" остальные признаки.
  + **Зеленые треугольники вверх (Lasso alpha=0.01):** Больше точек имеют ненулевые коэффициенты (33 штуки), но **очень много** точек все еще **точно на линии Y=0**. Lasso провел отбор признаков, но менее жесткий.
  + **Пурпурные треугольники вниз (Lasso alpha=0.0001):** Почти все точки **не на нулевой линии**. Картина похожа на Ridge или OLS. Отбора признаков почти нет.
* **Главный вывод из этого графика:** Он **визуально доказывает** главную особенность Lasso: способность **обнулять коэффициенты** (точки ложатся точно на Y=0), в отличие от Ridge, который только уменьшает их (точки около Y=0, но не на ней). Это и есть **отбор признаков** в действии.

Надеюсь, теперь связь между кодом, числовыми результатами и визуальным представлением на графиках стала полностью ясна!

Окей, попробуем изложить ту же теорию проще и по пунктам.

**Линейные модели для Классификации: Проще говоря**

1. **Что делаем?**
   * Не предсказываем число (как в регрессии), а **делим объекты на группы (классы)**.
   * Начинаем с **двух** классов (бинарная классификация).
2. **Как делим?**
   * Представьте, что у вас есть точки двух цветов на плоскости.
   * Линейные модели пытаются провести **прямую линию**, чтобы отделить один цвет от другого.
   * Если признаков больше двух, то это будет **плоскость** или "гиперплоскость" (в многомерном пространстве).
   * Эта линия/плоскость называется **границей принятия решений**.
3. **Как работает формула?**
   * Формула похожа на регрессию: результат = вес1\*признак1 + вес2\*признак2 + ... + b
   * Но теперь мы просто смотрим на **знак** этого результата:
     + Результат > 0 ? Относим к классу **+1**.
     + Результат < 0 ? Относим к классу **-1**.
   * Граница принятия решений – это там, где результат **равен нулю**.
4. **Как найти лучшую линию? (Обучение)**
   * Есть разные методы (алгоритмы). Два самых частых:
     + **Логистическая регрессия** (LogisticRegression) - (Да, "регрессия" в названии, но это для **классификации**!)
     + **Линейный SVM** (LinearSVC)
   * Они немного по-разному решают, какая линия "лучшая".
5. **Проблема "слишком сложной" линии (Переобучение)**
   * Иногда модель проводит слишком извилистую (в многомерии) границу, чтобы идеально разделить точки на *обучающих* данных. Но такая граница плохо работает на *новых* данных.
   * Решение: **Регуляризация** – не даем модели стать слишком сложной.
6. **Регуляризация (Как не дать усложнить)**
   * Идея: Штрафовать модель за слишком большие веса (w). Модель старается держать веса поменьше.
   * **Параметр C (Очень важно!)**: Контролирует силу регуляризации.
     + **Большое C = СЛАБАЯ регуляризация**. Модель сильно старается правильно разделить *все* обучающие точки. Может переобучиться.
     + **Маленькое C = СИЛЬНАЯ регуляризация**. Модель ищет более простую, "гладкую" границу, может проигнорировать отдельные точки. Помогает от переобучения.
     + **Внимание:** C работает **наоборот** по сравнению с alpha в регрессии (где большое alpha = сильная регуляризация).
   * **Типы регуляризации:**
     + **L2** (по умолчанию): Просто делает веса меньше.
     + **L1** (можно включить): Тоже делает веса меньше, но некоторые может сделать **равными нулю**. Это помогает **выбрать самые важные признаки**.

**Коротко:** Линейные классификаторы проводят "разделительную черту" между классами. LogisticRegression и LinearSVC делают это с помощью регуляризации (настраивается параметром C), чтобы линия не была слишком сложной и хорошо работала на новых данных. Большое C = слабее контроль, маленькое C = сильнее контроль.

Отлично, давайте погрузимся в код и графики для линейной классификации.

**Шаг 1: Визуализация Границ Решений (LinearSVC и LogisticRegression)**

# Импорт библиотек

import mglearn

import sklearn

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression # Модель Логистической Регрессии

from sklearn.svm import LinearSVC # Модель Линейного SVM

# Загрузка простого 2D датасета 'forge' (2 признака, 2 класса)

X, y = mglearn.datasets.make\_forge()

# Создаем область для рисования: 1 ряд, 2 колонки для графиков

fig, axes = plt.subplots(1, 2, figsize=(10, 3))

# Проходимся по обеим моделям (LinearSVC и LogisticRegression) и их осям для рисования

for model, ax in zip([LinearSVC(), LogisticRegression()], axes):

# 1. Обучаем модель на данных X, y

clf = model.fit(X, y)

# 2. Рисуем РАЗДЕЛЯЮЩУЮ ЛИНИЮ (границу решений) для обученной модели 'clf'

# fill=False - не заливать области цветом

# eps=0.5 - для корректного отображения границы

mglearn.plots.plot\_2d\_separator(clf, X, fill=False, eps=0.5,

ax=ax, alpha=.7) # alpha - прозрачность линии

# 3. Рисуем сами ТОЧКИ данных (X[:, 0]-первый признак, X[:, 1]-второй, y-цвет/форма)

mglearn.discrete\_scatter(X[:, 0], X[:, 1], y, ax=ax)

# 4. Устанавливаем заголовок графика - имя модели

ax.set\_title("{}".format(clf.\_\_class\_\_.\_\_name\_\_))

# 5. Подписываем оси

ax.set\_xlabel("Признак 0")

ax.set\_ylabel("Признак 1")

# Добавляем легенду к первому графику (axes[0]) для обозначения классов

axes[0].legend()

plt.show() # Показать готовую фигуру с двумя графиками

**Объяснение Графика 1:**

* **Что на графике?** Вы видите два графика рядом. Левый - для LinearSVC, правый - для LogisticRegression. На каждом графике есть точки двух видов (синие круги и оранжевые треугольники) и одна прямая линия.
* **Оси:** Горизонтальная ось - "Признак 0", вертикальная - "Признак 1". Это два признака из нашего простого набора данных forge.
* **Точки:** Синие круги и оранжевые треугольники - это наши **данные**. Каждая точка - это один объект с двумя признаками. Цвет/форма показывают, к какому из **двух классов** (0 или 1) принадлежит объект.
* **Линия:** Это самое важное! Это **граница принятия решений**, которую нашла модель.
  + Модель считает, что все точки, лежащие **выше** этой линии, относятся к одному классу (например, оранжевые треугольники).
  + Все точки **ниже** линии - к другому классу (синие круги).
* **Сравнение:** Обратите внимание, что линии, найденные LinearSVC и LogisticRegression, очень похожи. Обе модели являются линейными классификаторами и по умолчанию используют L2 регуляризацию, поэтому на таких простых данных они часто дают схожие результаты.
* **Ошибки:** Посмотрите внимательно на точки. Видно, что обе модели **неправильно** классифицируют пару точек (например, синий круг попадает в оранжевую зону или наоборот). Это нормально, так как модели ищут *наилучшую* прямую линию, но не всегда возможно идеально разделить классы прямой.
* **Главный вывод из этого графика:** Линейные классификаторы (LinearSVC, LogisticRegression) разделяют классы с помощью прямой линии (границы принятия решений) в пространстве признаков.

**Шаг 2: Визуализация Влияния Регуляризации (C) на LinearSVC**

# --- ГРАФИК 2 ---

# Эта функция из mglearn специально показывает, как меняется граница решений

# LinearSVC при разных значениях параметра регуляризации C

mglearn.plots.plot\_linear\_svc\_regularization()

plt.show() # Показать график

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

IGNORE\_WHEN\_COPYING\_END

**Объяснение Графика 2:**

* **Что на графике?** Вы видите три графика рядом. Они показывают ту же самую задачу классификации (данные forge), но решенную LinearSVC с **разными значениями параметра C**.
* **Оси и Точки:** Такие же, как на Графике 1 (Признак 0, Признак 1, синие/оранжевые точки).
* **Линия (Граница Решений) - Ключевое отличие:**
  + **Левый график (C=0.01): Маленькое C = СИЛЬНАЯ регуляризация.**
    - Посмотрите на линию: она почти горизонтальная. Модель сильно "оштрафована" за сложность, поэтому она выбирает очень простую границу.
    - Она меньше подстраивается под отдельные точки и больше ориентируется на общую картину "большинства" точек.
    - Как результат, она неправильно классифицирует больше точек (несколько синих кругов и оранжевых треугольников).
  + **Центральный график (C=1.0): Значение C по умолчанию.**
    - Это тот же результат, что мы видели на Графике 1 для LinearSVC.
    - Линия наклонена сильнее, чем при C=0.01. Она лучше соответствует данным, но все еще допускает пару ошибок. Это компромисс между простотой и точностью на обучении.
  + **Правый график (C=100.0): Большое C = СЛАБАЯ регуляризация.**
    - Модель почти не штрафуется за сложность. Она изо всех сил **пытается правильно классифицировать каждую точку**.
    - Посмотрите, как сильно наклонилась линия! Она специально изменила наклон, чтобы правильно классифицировать те две точки, которые были ошибками при C=1.0.
    - На *этих* данных это выглядит хорошо, но на более сложных данных такое сильное подстраивание под обучающие точки может привести к **переобучению** (плохой работе на новых данных).
* **Главный вывод из этого графика:** Параметр C контролирует баланс между простотой модели и точностью на обучающих данных. **Маленькое C** -> сильная регуляризация -> **простая модель**, может недообучиться. **Большое C** -> слабая регуляризация -> **сложная модель**, старается все точки классифицировать верно, может переобучиться.

**Шаг 3: Логистическая Регрессия на Данных Рака Груди (Числовые Результаты)**

# --- РАБОТА С ДАТАСЕТОМ BREAST CANCER ---

from sklearn.datasets import load\_breast\_cancer # Загрузка датасета

cancer = load\_breast\_cancer() # В 'cancer' теперь данные и метки

from sklearn.model\_selection import train\_test\_split # Для разделения

# Делим данные: stratify=cancer.target важно для классификации,

# чтобы сохранить пропорции классов в обучении и тесте

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

cancer.data, cancer.target, stratify=cancer.target, random\_state=42)

# --- ОБУЧЕНИЕ И ОЦЕНКА LogisticRegression С РАЗНЫМИ C ---

# 1. C=1 (по умолчанию)

logreg = LogisticRegression(max\_iter=5000).fit(X\_train, y\_train) # Увеличим max\_iter на всякий случай

print("Правильность на обучающем наборе (C=1): {:.3f}".format(logreg.score(X\_train, y\_train)))

print("Правильность на тестовом наборе (C=1): {:.3f}".format(logreg.score(X\_test, y\_test)))

# 2. C=100 (СЛАБАЯ регуляризация)

logreg100 = LogisticRegression(C=100, max\_iter=5000).fit(X\_train, y\_train)

print("Правильность на обучающем наборе (C=100): {:.3f}".format(logreg100.score(X\_train, y\_train)))

print("Правильность на тестовом наборе (C=100): {:.3f}".format(logreg100.score(X\_test, y\_test)))

# 3. C=0.01 (СИЛЬНАЯ регуляризация)

logreg001 = LogisticRegression(C=0.01, max\_iter=5000).fit(X\_train, y\_train)

print("Правильность на обучающем наборе (C=0.01): {:.3f}".format(logreg001.score(X\_train, y\_train)))

print("Правильность на тестовом наборе (C=0.01): {:.3f}".format(logreg001.score(X\_test, y\_test)))

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

IGNORE\_WHEN\_COPYING\_END

**Объяснение Вывода 3, 4, 5 (Графика здесь нет):**

* **Что делает код:** Мы загрузили более сложный датасет breast\_cancer (много признаков) и обучили на нем LogisticRegression с тремя разными значениями C. Затем мы измерили точность (accuracy, т.к. score для классификаторов по умолчанию возвращает точность) на обучающем и тестовом наборах.
* **Результаты в выводе:**
  + **C=1:** Обучение: 0.955, Тест: 0.958. Точность высокая и **очень близкая** на обучении и тесте. Это хороший знак, модель хорошо обобщает, возможно, легкое недообучение (т.к. нет разрыва).
  + **C=100:** Обучение: 0.972, Тест: 0.965. Точность на обучении **выросла** (модель лучше подогналась из-за слабой регуляризации). Точность на тесте тоже немного выросла. Разрыв между ними чуть увеличился, но все еще мал. Эта модель чуть сложнее и чуть лучше работает.
  + **C=0.01:** Обучение: 0.934, Тест: 0.930. Точность на обучении **упала** (сильная регуляризация мешает подгонке). Точность на тесте тоже упала. Модель стала слишком простой (**недообучение**).
* **Главный вывод из этих чисел:** Мы видим, как C влияет на баланс. C=1 и C=100 дают хорошие результаты, причем C=100 чуть лучше подогнал данные. C=0.01 дает слишком простую модель. Для этого датасета слабая регуляризация (C=100) или регуляризация по умолчанию (C=1) работают хорошо.

**Шаг 4: Визуализация Коэффициентов LogisticRegression (L2 регуляризация)**

# --- ГРАФИК 3 ---

# Рисуем коэффициенты для трех моделей LogisticRegression с L2 регуляризацией

plt.plot(logreg.coef\_.T, 'o', label="C=1") # Коэффициенты для C=1 (круги)

plt.plot(logreg100.coef\_.T, '^', label="C=100") # Коэффициенты для C=100 (треугольники вверх)

plt.plot(logreg001.coef\_.T, 'v', label="C=0.001") # Ошибка в тексте? Должно быть C=0.01. Используем переменную logreg001. (треуг. вниз)

# Делаем подписи оси X именами признаков из датасета cancer, поворачиваем их на 90 град.

plt.xticks(range(cancer.data.shape[1]), cancer.feature\_names, rotation=90)

plt.hlines(0, 0, cancer.data.shape[1]) # Горизонтальная линия на нуле

plt.ylim(-5, 5) # Ограничение оси Y

plt.xlabel("Индекс коэффициента") # На самом деле теперь это Имя Признака

plt.ylabel("Оценка коэффициента")

plt.legend() # Показать легенду

plt.show() # Показать график

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

IGNORE\_WHEN\_COPYING\_END

**Объяснение Графика 3:**

* **Что на графике?** График, показывающий "важность" каждого признака для трех моделей LogisticRegression с разными C.
* **Оси:**
  + Горизонтальная ось (X): **Имена признаков** датасета breast\_cancer (mean radius, mean texture и т.д.).
  + Вертикальная ось (Y): **Величина коэффициента w**, который модель присвоила этому признаку. Чем дальше коэффициент от нуля (вверх или вниз), тем важнее этот признак для модели.
* **Точки (Коэффициенты):**
  + **Синие круги (C=1):** Коэффициенты для модели по умолчанию. Мы видим, что некоторые признаки имеют довольно большие положительные или отрицательные коэффициенты.
  + **Оранжевые треугольники вверх (C=100): Большое C = СЛАБАЯ регуляризация.** Коэффициенты в целом стали **еще больше** по модулю, чем при C=1. Модель придает большее значение некоторым признакам, она сложнее.
  + **Зеленые треугольники вниз (C=0.01): Маленькое C = СИЛЬНАЯ регуляризация.** Посмотрите, все коэффициенты стали **очень маленькими**, они сильно "сжаты" к нулю. Модель считает почти все признаки малозначимыми из-за сильного штрафа.
* **Горизонтальная линия Y=0:** Уровень нулевой важности признака.
* **Главный вывод из этого графика:** L2 регуляризация (которая используется здесь по умолчанию) **уменьшает величину коэффициентов**. Сила этого уменьшения зависит от C: маленькое C сильно сжимает коэффициенты к нулю (делая модель проще), большое C позволяет коэффициентам быть больше (модель сложнее). Но L2 **редко делает коэффициенты точно равными нулю**.

**Шаг 5: Визуализация Коэффициентов LogisticRegression (L1 регуляризация - *ожидаемая*)**

# --- ГРАФИК 4 ---

# ВНИМАНИЕ: В предоставленном коде здесь ошибка, используется penalty="l2" вместо "l1".

# Я объясню код как есть (L2), а потом как он должен был бы работать с L1.

# Цикл по значениям C и маркерам для графика

for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):

# Обучаем модель. В коде ОШИБКА: стоит penalty="l2", должно быть "l1"

lr\_l1 = LogisticRegression(C=C, penalty="l2", solver='liblinear', max\_iter=10000).fit(X\_train, y\_train) # Добавил solver и max\_iter

# Вывод точности для текущей модели (которая на самом деле L2)

print("Правильность на обучении для логрегрессии l2 с C={:.3f}: {:.2f}".format( C, lr\_l1.score(X\_train, y\_train)))

print("Правильность на тесте для логрегрессии l2 с C={:.3f}: {:.2f}".format( C, lr\_l1.score(X\_test, y\_test)))

# Рисуем коэффициенты текущей модели (которая L2)

plt.plot(lr\_l1.coef\_.T, marker, label="L2 C={:.3f}".format(C)) # Подписал как L2

# Настройки графика (такие же, как для графика 3)

plt.xticks(range(cancer.data.shape[1]), cancer.feature\_names, rotation=90)

plt.hlines(0, 0, cancer.data.shape[1])

plt.xlabel("Индекс коэффициента")

plt.ylabel("Оценка коэффициента")

plt.ylim(-5, 5)

plt.legend(loc=3) # Легенда внизу слева

plt.show()

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487).Python

IGNORE\_WHEN\_COPYING\_END

**Объяснение Графика 4 (Как работает код С ОШИБКОЙ - L2):**

* **Что на графике?** Этот график, из-за ошибки penalty="l2" в коде, будет выглядеть **практически идентично Графику 3**. Он снова показывает эффект L2 регуляризации с разными C. Вы увидите сжатие коэффициентов к нулю, но не их обнуление.
* **Вывод из этого графика (с ошибкой):** Повторяет выводы Графика 3 о влиянии C на L2 регуляризацию.

**Объяснение Графика 4 (Как ДОЛЖЕН БЫЛ работать код с L1):**

* **Что мы ожидали бы увидеть, если бы было penalty="l1"?**
* **Оси:** Те же (Имена признаков, Значение коэффициента).
* **Точки (Коэффициенты с L1):**
  + **Круги (C=0.001): Маленькое C = СИЛЬНАЯ L1 регуляризация.** Мы бы увидели, что **большинство** коэффициентов стали **ТОЧНО РАВНЫ НУЛЮ**. Только очень немногие признаки имели бы ненулевые (скорее всего, маленькие) коэффициенты. График был бы очень "разреженным" (sparse). Это **отбор признаков**.
  + **Треугольники вверх (C=1):** L1 регуляризация слабее. Ненулевых коэффициентов было бы **больше**, чем при C=0.001, но все равно **значительно меньше**, чем общее число признаков. Отбор признаков все еще заметен.
  + **Треугольники вниз (C=100): Большое C = СЛАБАЯ L1 регуляризация.** Большинство (возможно, почти все) коэффициенты были бы **ненулевыми**. Картина была бы больше похожа на L2 регуляризацию (График 3), но все еще возможно, что некоторые наименее важные признаки были бы обнулены.
* **Главный вывод (для L1):** L1 регуляризация в LogisticRegression (при использовании penalty="l1") приводит к **обнулению** коэффициентов для менее важных признаков. Это особенно заметно при сильной регуляризации (маленьком C). L1 выполняет **автоматический отбор признаков**, делая модель проще и потенциально более интерпретируемой.

Надеюсь, этот подробный разбор кода и графиков помог!