Интеллектуальный анализ данных на основе методов машинного обучения

Скобцов Вадим Юрьевич
К.т.н., доцент
+7 981 023-99-58 (WhatsApp, telegram)
E-mail: vasko_vasko@mail.ru

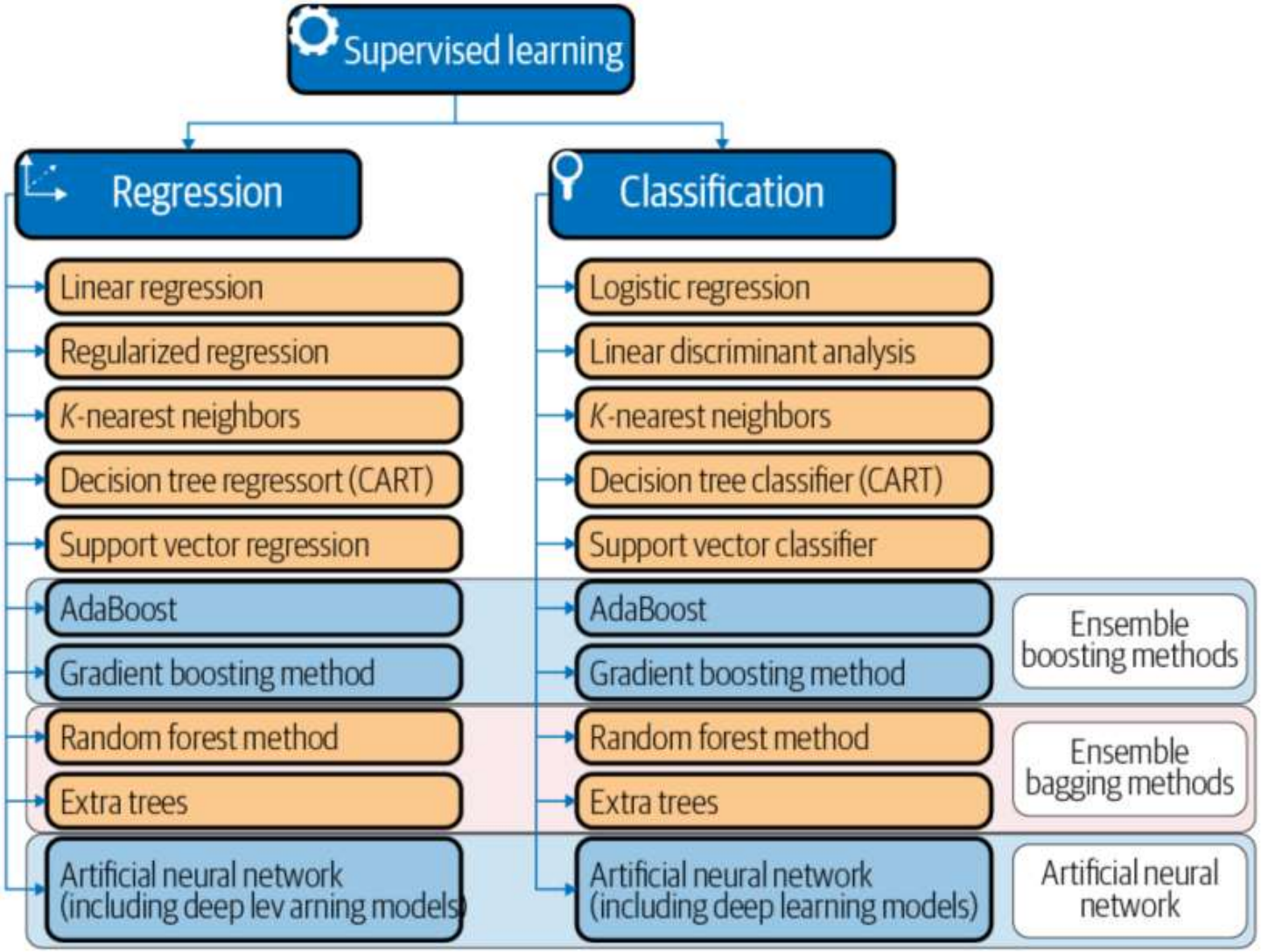
Раздел I. Машинное/глубокое обучение как современная методология интеллектуального анализа данных. Тема I.3. Основные метрики оценки моделей машинного/глубокого обучения.

1 Обучение с учителем

- Обучение с учителем — это область машинного обучения, в которой выбранный алгоритм пытается обучиться эталонной цели, используя заданные входные данные. Алгоритму передается набор обучающих данных, содержащий целевые эталонные метки. Основываясь на обучающем наборе данных, алгоритм обучается правилу, которое он далее использует для прогнозирования меток для новых входных наблюдений. Другими словами, алгоритмы обучения с учителем получают исторические обучающие данные, на основе которых строится предиктивная или классификационная модель с наилучшим заданным качеством.
- Как говорилось выше, существует две разновидности алгоритмов обучения с учителем: алгоритмы регрессии и классификации. Методы обучения с учителем на основе регрессии пытаются предсказать результаты на основе входных переменных. Методы обучения с учителем на основе классификации определяют, к какой категории относится набор элементов данных. Алгоритмы классификации являются вероятностными, что означает, что результатом является категория, для которой алгоритм находит наивысшую вероятность того, что элемент набора данных принадлежит к ней. Алгоритмы регрессии, напротив, оценивают результат задач, которые имеют бесконечное число решений (непрерывный набор возможных результатов).

1 Обучение с учителем

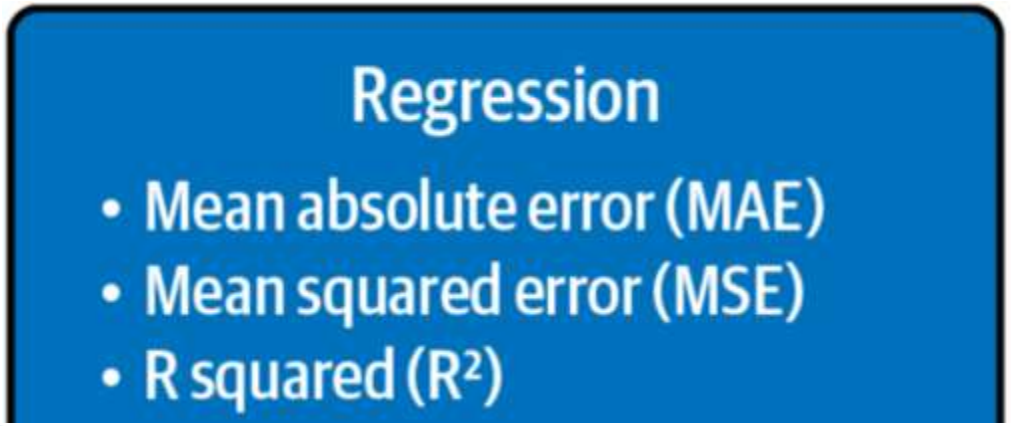
- Проблемы моделирования прогностической (предиктивной) классификации отличаются от задач моделирования прогностической (предиктивной) регрессии, поскольку классификация — это задача прогнозирования метки дискретного класса, а регрессия — задача прогнозирования непрерывной величины. Тем не менее, обе модели используют одну и ту же концепцию использования известных переменных для прогнозирования, и между этими двумя моделями есть значительное совпадение. На рисунке обобщается список моделей, обычно используемых для классификации и регрессии. Некоторые модели можно использовать как для классификации, так и для регрессии с небольшими изменениями. Это алгоритмы K-ближайших соседей, деревьев принятия решений, опорных вектор, ансамблевые методы бэггинга/бустинга и искуственные нейронные сети (включая глубокие нейронные сети).



1 Обучение с учителем

1.1 Метрики оценки моделей регрессии

- Выбор метрики для оценки качества алгоритмов машинного обучения является очень важным. Существенным аспектом показателей оценки является способность различать результаты модели. Различные типы метрик оценки используются для разных моделей машинного обучения.
- Выбор используемых показателей влияет на то, как измеряется и сравнивается качество алгоритмов машинного обучения. Метрики влияют как на то, как вы оцениваете важность различных характеристик в результатах, так и на ваш окончательный выбор алгоритма.
- Ниже приведены основные метрики моделей МО для случаев регрессии. На метриках регрессионных моделей мы остановимся подробнее. Случай классификационных метрик рассмотрим ниже, когда дойдем до моделей классификации.



1 Обучение с учителем

1.1 Метрики оценки моделей регрессии

- Введем следующие обозначения:
  - m – объем обучающего набора данных – количество образцов в наборе данных
  - $x^{(i)}$ – вектор всех входных значений признаков *i*–го образца в наборе данных – независимые входные переменные;
  - $y^{(i)}$ – его метка (целевое эталонное выходное значение для данного образца) – зависимая выходная переменная;
  - $(x^{(i)}, y^{(i)})$ – *i*–й элемент обучающего набора данных;
  - $\boldsymbol{X}$ – матрица, содержащая все элементы набора данных, предусмотрена одна строка на образец, $\boldsymbol{Y}$ вектор–столбец меток – целевых эталонных значений модели регрессии;
  - h – функция модели прогнозирования системы, также называемая гипотезой (hypothesis), когда модели подается вектор признаков образца $x^{(i)}$, она выводит для этого образца прогнозируемое значение $\hat{y}^{(i)} = h(x^{(i)})$.

1 Обучение с учителем

1.1 Метрики оценки моделей регрессии

Постановка задачи регрессии

- ***Регрессия*** — это линейная/нелинейная модель, которая предполагает линейную/нелинейную связь между входными переменными $x = (x_1, x_2, \ldots, x_n)$ и единственной выходной переменной y. Цель регрессии состоит в том, чтобы научить модель предсказывать новый y по ранее неизвестному x с минимально возможной ошибкой.
- Как правило, линейная модель вырабатывает прогноз, просто вычисляя взвешенную сумму входных признаков плюс константы под названием *член смещения* (*bias term*), также называемой *свободным членом* (*intercept term*), как показано в уравнении

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = h_\theta(\mathbf{x}) = \theta^T \cdot \mathbf{x}$$

Где

- $\hat{y}$ — спрогнозированное значение.
- n — количество признаков.
- x_i — значение i-того признака.
- θ — *вектор параметров* модели, содержащий член смещения θ_0 и веса признаков от θ_1 до θ_n.
- θ^T — транспонированный θ (вектор-строка вместо вектора-столбца).
- $\mathbf{x}$ — *вектор признаков* образца, содержащий от x_0 до x_n, где x_0 всегда равно 1.
- $\theta^T \cdot \mathbf{x}$ — скалярное произведение θ^T и $\mathbf{x}$.
- h_θ — функция гипотезы, использующая параметры модели θ.

1 Обучение с учителем

1.1 Метрики оценки моделей регрессии

- *Среднеквадратическая ошибка* (Mean Squared Error) MSE(**X**,*h*) и *средняя абсолютная ошибка* (Mean Absolute Error) MAE(**X**,*h*) — это функции стоимости, измеренные на наборе данных с использованием значений *h* вашей модели МО по сравнению с эталонными целевыми значениями обучения *y*.
- И MSE, и MAE — это способы измерения расстояния между двумя векторами: вектором прогнозов и вектором целевых эталонных значений.
- **Средняя абсолютная ошибка**
  Средняя абсолютная ошибка (MAE) представляет собой среднюю сумму абсолютных разностей между прогнозируемыми и целевыми значениями.

$$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right| \qquad (2.1)$$

- MAE является линейной оценкой, что означает, что все индивидуальные различия имеют одинаковый вес в среднем. Это дает представление о том, насколько ошибочными были прогнозы.
- Мера дает представление о величине ошибки, но не дает представления о направлении (например, завышенное или заниженное предсказание).
- В пакете scikit-learn есть функция, реализующая эту метрику:

sklearn.metrics.**mean_absolute_error**(*y_true*, *y_pred*, *, *sample_weight=None*, *multioutput='uniform_average'*)
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html
https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

1 Обучение с учителем

1.1 Метрики оценки моделей регрессии

- **Среднеквадратическая ошибка**
  Среднеквадратическая ошибка (MSE) представляет собой выборочное стандартное отклонение различий между прогнозируемыми значениями и наблюдаемыми значениями (называемые остатками).
- Это очень похоже на среднюю абсолютную ошибку, поскольку дает общее представление о величине ошибки.

$$\mathrm{MSE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2 \qquad (2.2)$$

- Извлечение квадратного корня из среднеквадратичной ошибки преобразует единицы обратно в исходные единицы выходной переменной и может иметь смысл для описания и представления. Это называется корнем квадратным из среднеквадратичной ошибкой (root mean squared error – RMSE).

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2} \qquad (2.3)$$

- В пакете scikit-learn есть функция, реализующая эту метрику:

sklearn.metrics.**mean_squared_error**(*y_true*, *y_pred*, *, *sample_weight=None*, *multioutput='uniform_average'*, *squared=True*)
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

1 Обучение с учителем

1.1 Метрики оценки моделей регрессии

- Существуют разнообразные меры расстояния, или нормы.

- Вычисление квадратного корня из суммы квадратов (RMSE) соответствует *евклидовой норме* (*euclidian norm*): это понятие расстояния, с которым вы знакомы. Она также называется *нормой* ℓ_2 и обозначается как $\|\cdot\|_2$ (или просто $\|\cdot\|$).
- Вычисление суммы абсолютных величин (MAE) соответствует *норме* ℓ_1 и обозначается как $\|\cdot\|_1$. Иногда ее называют *нормой Манхэттена* (*Manhattan norm*), потому что она измеряет расстояние между двумя точками в городе, если вы можете перемещаться только вдоль прямоугольных городских кварталов.

1 Обучение с учителем

1.1 Метрики оценки моделей регрессии

❑ Существуют разнообразные меры расстояния, или нормы.

- В более общем смысле *норма* ℓ_k вектора **v**, содержащего n элементов, определяется как $\| \mathbf{v} \|_k = (|v_0|^k + |v_1|^k + ... + |v_n|^k)^{\frac{1}{k}}$. ℓ_0 просто дает количество ненулевых элементов в векторе, а ℓ_∞ — максимальную абсолютную величину в векторе.

- Чем выше индекс нормы, тем больше она концентрируется на крупных значениях и пренебрегает мелкими значениями. Вот почему ошибка RMSE чувствительнее к выбросам, чем ошибка MAE. Но когда выбросы экспоненциально редкие (как в колоколообразной кривой), ошибка RMSE работает очень хорошо и обычно является предпочтительной.

1 Обучение с учителем

1.1 Метрики оценки моделей регрессии

- **R^2 метрика**

  R^2 метрика вычисляет коэффициент детерминации. R^2 метрика представляет собой долю дисперсии для y, которая была объяснена независимыми переменными в модели. Ее значение показывает «соответствие» прогнозов целевому значению и, следовательно, меру того, насколько хорошо скрытые выборки могут быть предсказаны моделью через долю объясненной дисперсии.

- *Определение*

  R^2 метрика определяется как:

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{m}(y^{(i)} - \bar{y})^2}, \qquad (2.4)$$

где $\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}$ и $\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2 = \sum_{i=1}^{m} e^{(i)^2}$.

- В пакете scikit-learn есть функция, реализующая эту метрику:

sklearn.metrics.**r2_score**(*y_true*, *y_pred*, ***, *sample_weight=None*, *multioutput='uniform_average'*, *force_finite=True*)

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html
https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

1 Обучение с учителем

1.1 Метрики оценки моделей регрессии

- **R^2 метрика**
- Поскольку такая дисперсия зависит от набора данных, R^2 не может быть осмысленно сопоставимым для разных наборов данных. Наилучшая возможная оценка — 1,0, и она может быть отрицательной (поскольку модель может быть сколь угодно хуже). Для приемлемых моделей предполагается, что коэффициент детерминации должен быть хотя бы не меньше 0.5 (в этом случае коэффициент множественной корреляции превышает по модулю 0.7). Модели с коэффициентом детерминации выше 0.8 можно признать достаточно хорошими (коэффициент корреляции превышает 0.9). Значение коэффициента детерминации 1 означает функциональную зависимость между переменными.
- Обратите внимание, что r2_score рассчитывается нескорректировано без поправки на систематическую ошибку выборочной дисперсии y. В частном случае, когда истинная цель постоянна, оценка не конечна: это либо NaN (идеальные прогнозы), либо -Inf (несовершенные прогнозы). Такие неконечные оценки могут помешать правильному выполнению правильной оптимизации модели, такой как перекрестная проверка поиска по сетке. По этой причине поведение r2_score по умолчанию заключается в замене их на 1,0 (идеальные прогнозы) или 0,0 (несовершенные прогнозы). Если для force_finite установлено значение False, эта оценка возвращается к исходному значению.

1 Обучение с учителем

1.2 Линейная регрессия

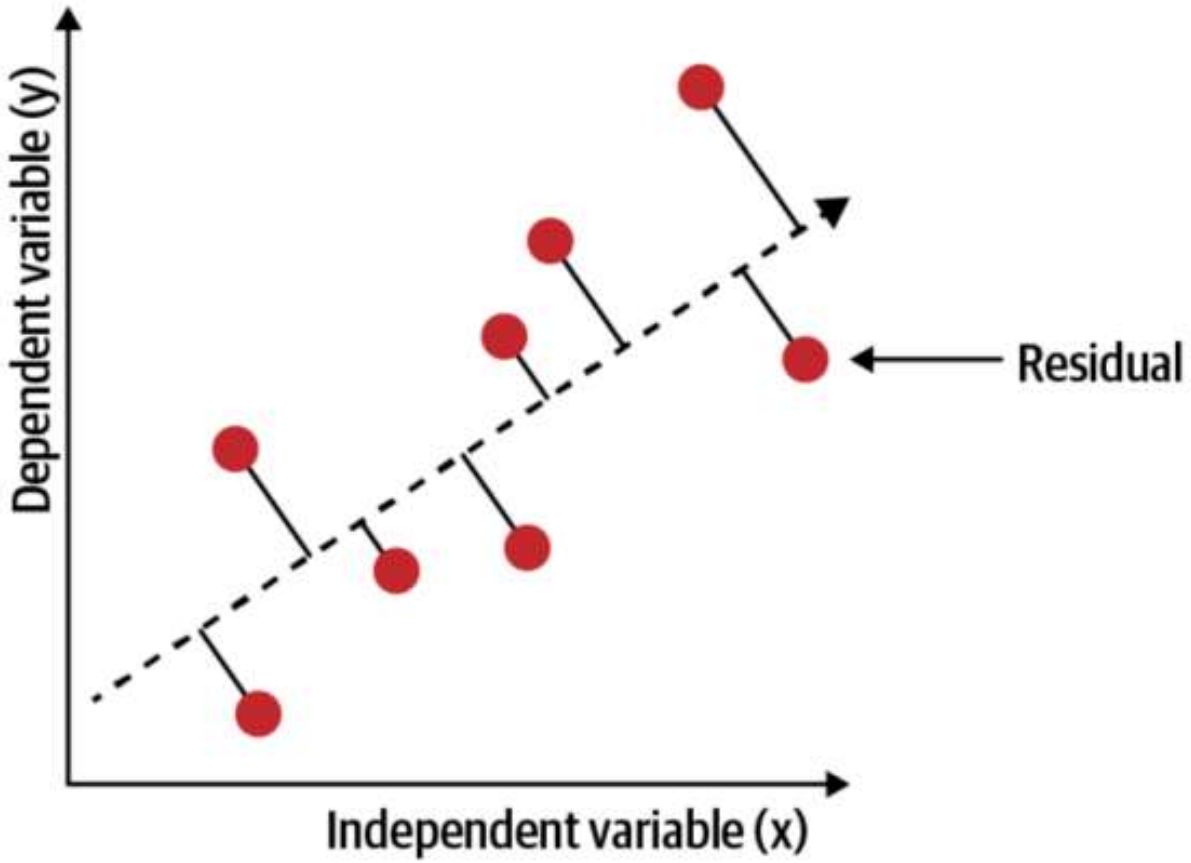
- ***Обучение модели линейной регрессии***
- *Мера качества модели*
- Обучение модели означает установку ее параметров $\boldsymbol{\theta}$ так, чтобы модель была наилучшим образом подогнана к обучающему набору. Для этой цели нам первым делом нужна мера того, насколько хорошо (или плохо) модель подогнана к обучающим данным.
- Ранее мы выяснили (слайды 6–9), что распространенной мерой производительности регрессионной модели является квадратный корень из среднеквадратической ошибки RMSE (формула (2.3)).
- Следовательно, для обучения линейной регрессионной модели необходимо найти значение $\boldsymbol{\theta}$, которое сводит к минимуму RMSE. На практике проще довести до минимума среднеквадратическую ошибку MSE, чем RMSE, что приведет к тому же самому результату (поскольку значение, которое сводит к минимуму функцию, также сводит к минимуму ее квадратный корень).
- Ошибка MSE гипотезы линейной регрессии h_θ на обучающем наборе $\boldsymbol{X}$ вычисляется так:

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{\theta}^T \cdot x^{(i)} - y^{(i)})^2 \qquad (2.5)$$

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- *Мера качества модели*
- По сути цель обучения модели линейной регрессии состоит в том, чтобы минимизировать остаточную сумму квадратов между целевыми эталонными значениями меток в обучающем наборе данных и значениями целевого признака, предсказанными линейным регрессионным приближением.



1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- ***Задача оптимизации для линейной регрессии, формальная постановка***

- Предлагается гипотеза на основе модели линейной регрессии $h_\theta(x)$
- Гипотеза определяется параметрами вектора $\boldsymbol{\theta}$
- Задана функция потерь

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{\theta}^T \cdot x^{(i)} - y^{(i)})^2$$

- Необходимо решить задачу минимизации функции потерь

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \qquad (2.6)$$

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- *Нормальное уравнение – обучение на основе решения в аналитическом виде*
- Для нахождения значения $\boldsymbol{\theta}$, которое сводит к минимуму функцию издержек, имеется *решение в аналитическом виде* (*closed-form solution*) – иными словами, математическое уравнение, дающее результат напрямую. Оно называется *нормальным уравнением* (*normal equation*) и представлено в уравнении:

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y} \qquad (2.7)$$

- $\hat{\theta}$ — значение θ, которое сводит к минимуму функцию издержек.
- $\mathbf{y}$ — вектор целевых значений, содержащий от $y^{(1)}$ до $y^{(m)}$.

- Решение получено методом наименьших квадратов (МНК).

https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BD%D0%B0%D0%B8%D0%BC%D0%B5%D0%BD%D1%8C%D1%88%D0%B8%D1%85_%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BE%D0%B2

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- *Нормальное уравнение – обучение на основе решения в аналитическом виде*
- Давайте сгенерируем данные, выглядящие как линейные, для проверки на этом уравнении:

```
#Linear dataset random generation

import numpy as np

X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

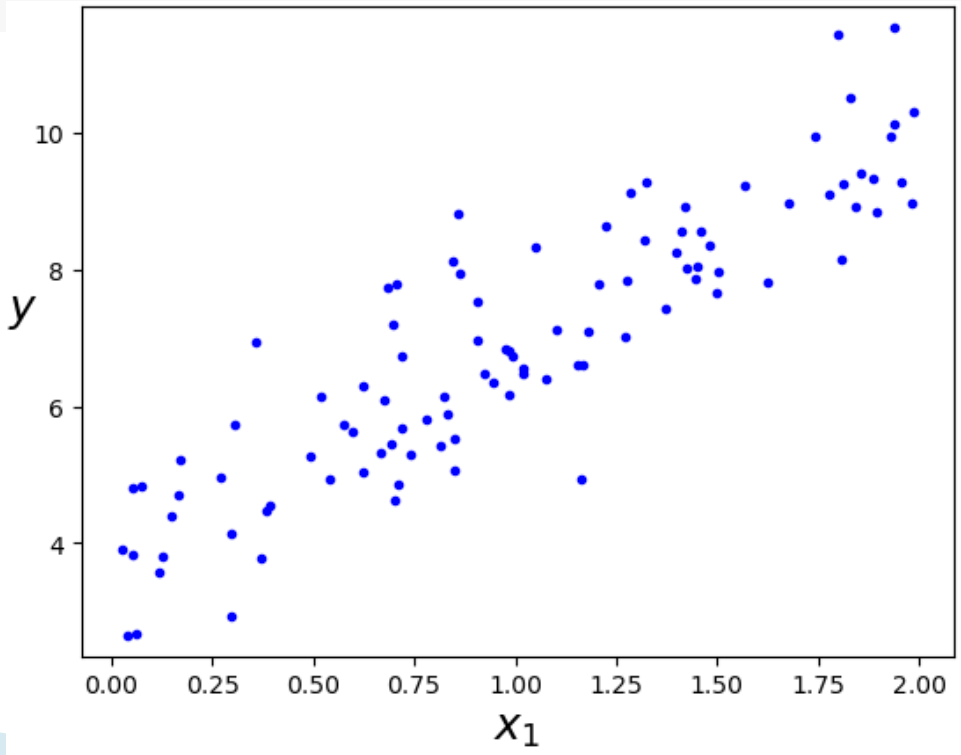
```
#Visualisation of dataset
import matplotlib.pyplot as plt
plt.plot(X, y, "b.")
plt.xlabel("$x$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([X[:].min()-0.1,X[:].max()+0.1, y[:].min()-0.3,y[:].max()+0.3])
plt.show()
```

```
X
```

```
array([[1.84467387],
       [0.05157557],
       [1.95630485],
       [1.56658613],
       [0.37239924],
       [0.66507303],
       [0.8159907 ],
       [1.42390523],
       [0.53991562],
       [0.30479623],
       [0.03998737],
```

```
y
```

```
array([[ 8.93253574],
       [ 3.8341326 ],
       [ 9.27119017],
       [ 9.24004811],
       [ 3.771767  ],
       [ 5.31849137],
       [ 5.42478309],
       [ 8.01462448],
       [ 4.93608968],
       [ 5.72220909],
       [ 2.64649811],
```



1 Обучение с учителем

1.2 Линейная регрессия

- *Обучение модели линейной регрессии*
- Эквивалентный код, в котором применяются функции пакета Scikit-Learn, выглядит так:

```
In [74]: from sklearn.linear_model import LinearRegression
         lin_reg = LinearRegression()
         lin_reg.fit(X, y)
         lin_reg.intercept_, lin_reg.coef_

Out[74]: (array([3.75066965]), array([[3.14047097]]))
```

```
y_predict = lin_reg.predict(X_new)
print("Значения прогноза линейной регрессии: \n", y_predict)
#Target y values
y_target= 4 + 3 * X_new
print("Целевые значения: \n", y_target)
rmse_lr = np.sqrt(mean_squared_error(y_target,y_predict))
r2_lr = r2_score(y_target,y_predict)
print("RMSE for LR:", rmse_lr)
print("R2_score for LR:", r2_lr)
```

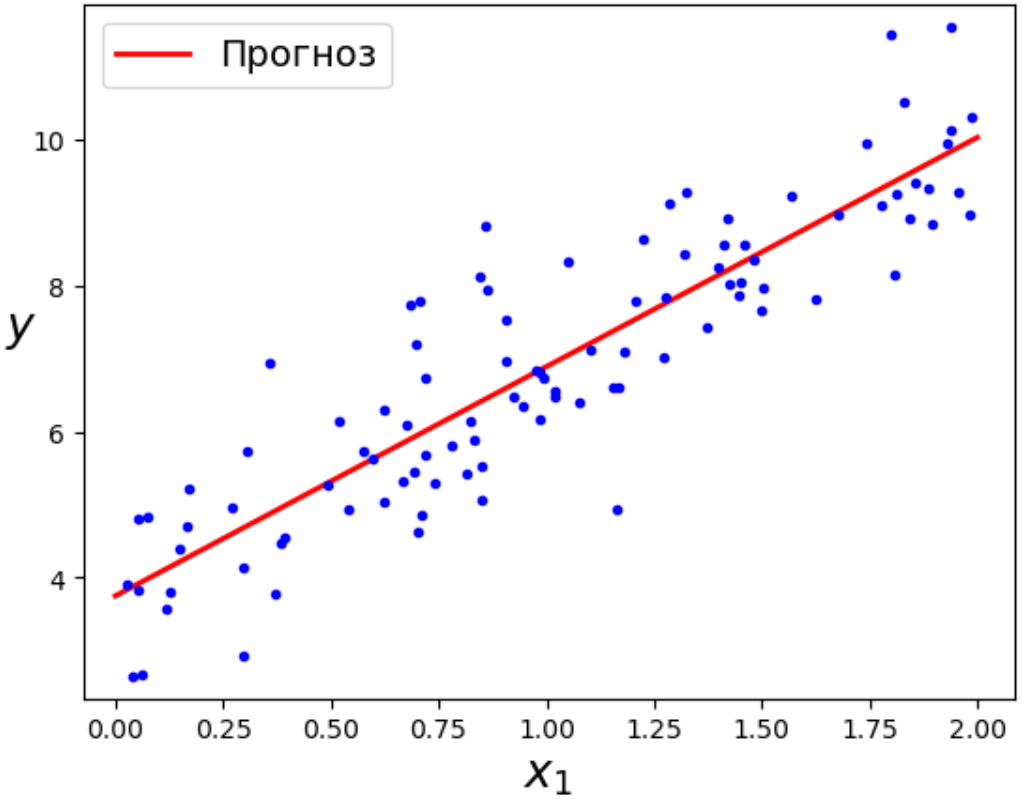
```
Значения прогноза линейной регрессии:
 [[ 3.75066965]
 [10.03161159]]
Целевые значения:
 [[ 4]
 [10]]
RMSE for LR: 0.17771454005498957
R2_score for LR: 0.996490838028116
```

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- Построим график полученной предиктивной модели линейной регрессии:

```
#Plotting the LR model graphic
import matplotlib.pyplot as plt
plt.plot(X_new, y_predict, "r-", linewidth=2, label="Прогноз")
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([X[:].min()-0.1,X[:].max()+0.1, y[:].min()-0.3,y[:].max()+0.3])
plt.show()
```



1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- *Вычислительная сложность обучения на основе решения в аналитическом виде*
- Нормальное уравнение вычисляет инверсию $\boldsymbol{X} \cdot \boldsymbol{X}^{\boldsymbol{T}}$, которая представляет собой матрицу $n \times n$ (где n – количество признаков). Вычислительная сложность инвертирования такой матрицы обычно составляет примерно от $O(n^{2.4})$ до $O(n^3)$ (в зависимости от реализации).
  Другими словами, если вы удвоите количество признаков, то умножите время вычислений на значение приблизительно от $2^{2.4} = 5.3$ до $2^3 = 8$.
  Обучение на основе аналитического решения нормального уравнения становится очень медленным, когда количество признаков серьезно возрастает (скажем, до 100000).
- В качестве положительной стороны следует отметить, что это уравнение является линейным относительно числа образцов в обучающем наборе ($O(m)$), а потому оно эффективно обрабатывает крупные обучающие наборы, если только они могут разместиться в памяти.

1 Обучение с учителем

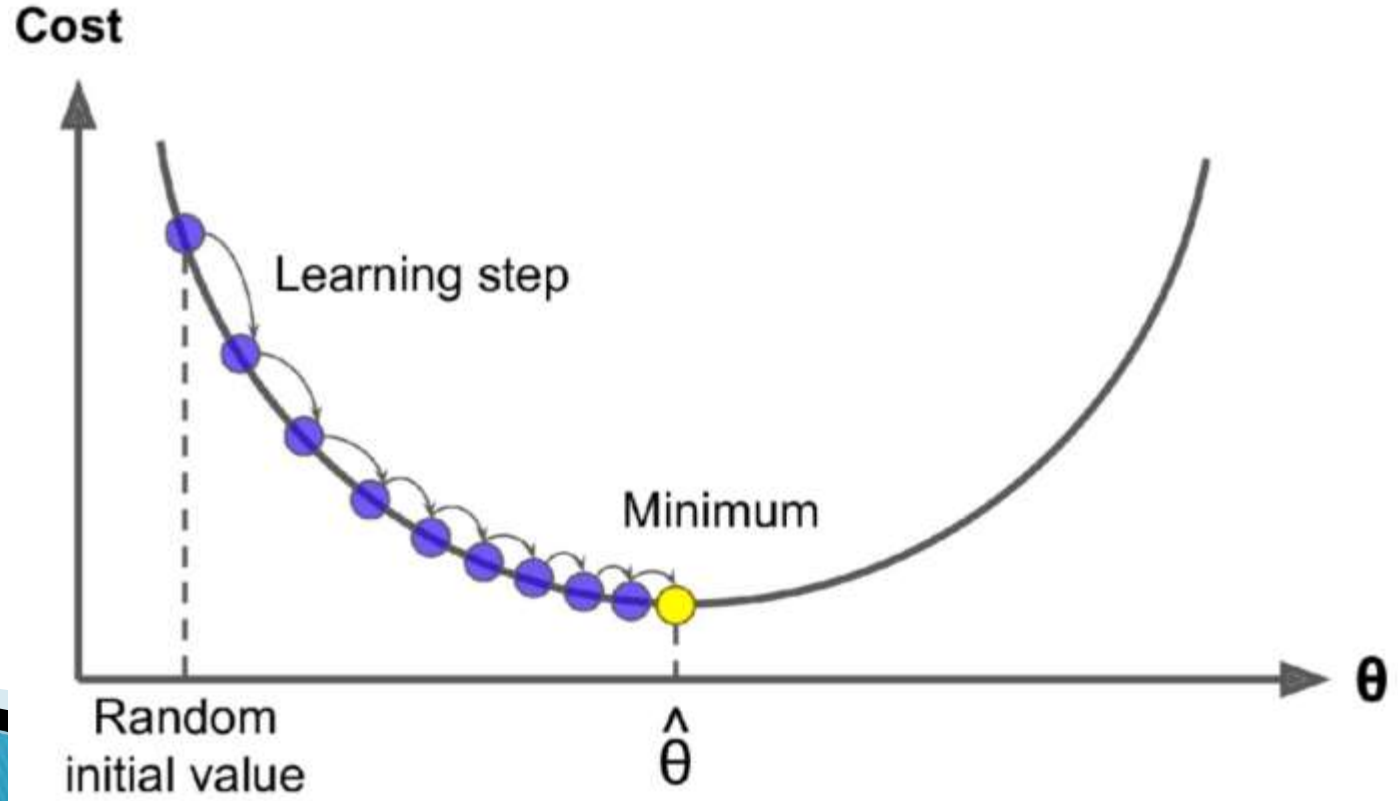
1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска***
  - Метод градиентного спуска представляет собой алгоритм оптимизации, способный находить оптимальные решения широкого диапазона задач.
  - Основная идея градиентного спуска заключается в том, чтобы итеративно подстраивать параметры для сведения к минимуму функции издержек $J(\boldsymbol{\theta})$.
  - Предположим, вы потерялись в горах в густом тумане и вы способны прощупывать только крутизну поверхности под ногами.
  - Хорошая стратегия быстро добраться до дна долины предусматривает движение вниз по самому крутому спуску (не поступайте так в реальных горах, потому что быстро – не значит безопасно). Это в точности то, что делает градиентный спуск: он измеряет локальный градиент функции ошибок применительно к вектору параметров $\boldsymbol{\theta}$ и двигается в направлении убывающего градиента. Как только градиент становится нулевым, вы достигли минимума.

1 Обучение с учителем

1.2 Линейная регрессия

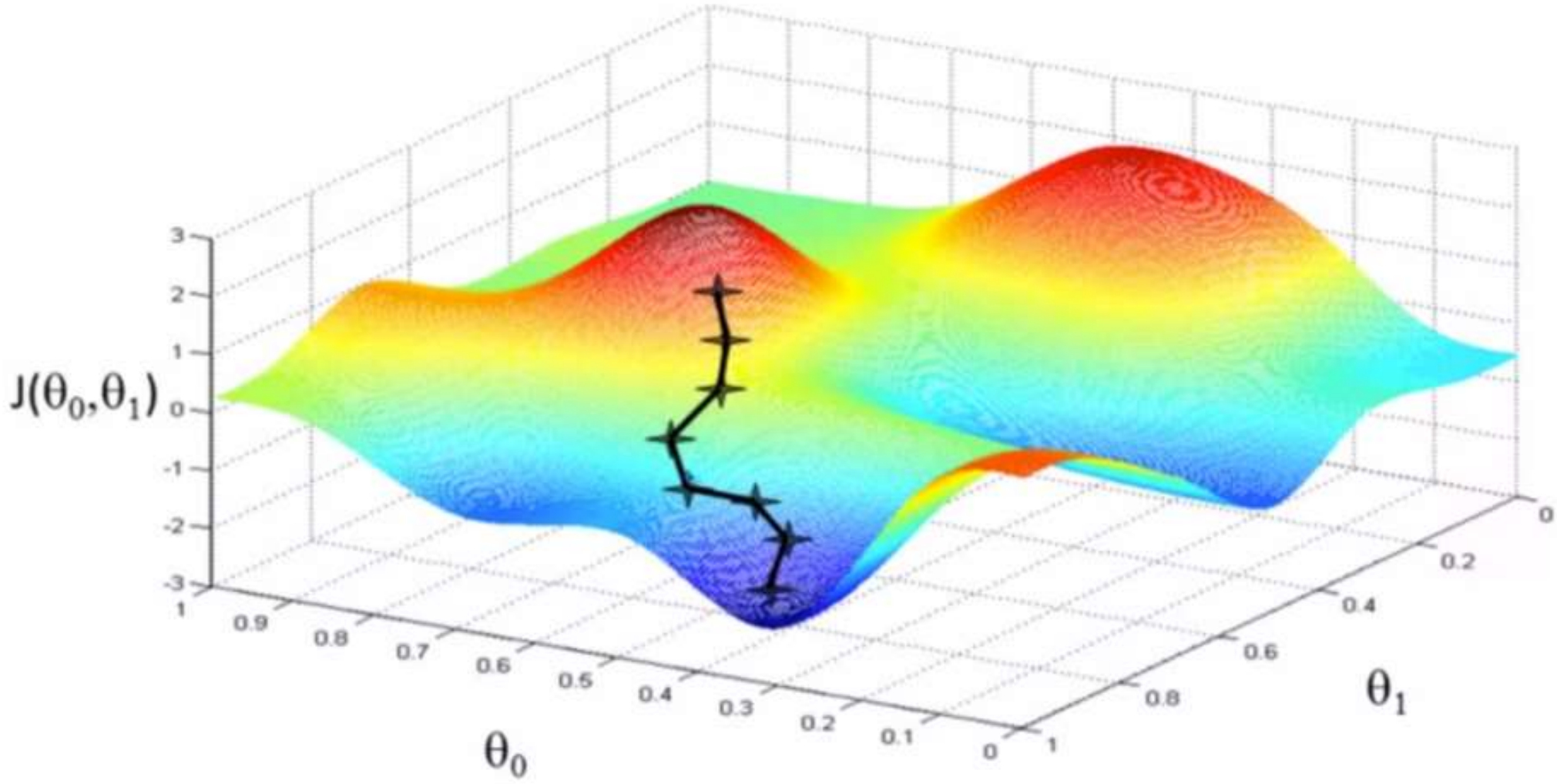
- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска***
  - Выражаясь более конкретно, вы начинаете с наполнения вектора $\boldsymbol{\theta}$ случайными значениями (т.н. случайная инициализация).
  - Затем вы постепенно улучшаете его, предпринимая по одному маленькому шагу за раз и на каждом шаге пытаясь снизить функцию издержек $J(\boldsymbol{\theta})$ (MSE) до тех пор, пока алгоритм не сойдется в минимуме достаточного уровня.



1 Обучение с учителем

1.2 Линейная регрессия

- *Обучение модели линейной регрессии*
  - *Обучение на основе метода градиентного спуска*



1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска***
  - Чтобы реализовать градиентный спуск, вам понадобится вычислить градиент функции издержек в отношении каждого параметра модели θ_j · Другими словами, вам необходимо подсчитать, насколько сильно функция издержек будет изменяться при небольшом изменении θ_j. Очевидно, что в результате мы получим вектор частных производных по параметру θ_j. Это похоже на то, как задать вопрос: "Каков угол наклона горы под моими ногами, если я стою лицом на восток?", затем задать его, повернувшись на север (и т.д. для всех остальных направлений или измерений).
    - Формула ниже вычисляет частную производную функции издержек в отношении параметра θ_j:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{\partial}{\partial \theta_j} \mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{\theta}^T \cdot x^{(i)} - y^{(i)}) x_j^{(i)} \tag{2.8}$$

  - Таким образом, алгоритм градиентного спуска минимизации функции потерь $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ выглядит так:

Пока не достигнута сходимость
{
  Для $j = \overline{0, n-1}$
  {

$$\theta_j = \theta_j - \eta \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{\theta}^T \cdot x^{(i)} - y^{(i)}) x_j^{(i)} \tag{2.9}$$

  }
}

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- ***Обучение на основе метода градиентного спуска. Пакетный градиентный спуск***
- Вместо вычисления таких частных производных по отдельности вы можете воспользоваться формулой (2.10), чтобы вычислить их все сразу. Вектор-градиент $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$, содержит все частные производные функции издержек (по одной для каждого параметра модели):

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\boldsymbol{\theta}) \\ \dots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \boldsymbol{X}^T (\boldsymbol{X} \cdot \boldsymbol{\theta} - \boldsymbol{y}) \tag{2.10}$$

- Таким образом, алгоритм градиентного спуска минимизации функции потерь $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ в векторной форме выглядит так:

Пока не достигнута сходимость

{

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \frac{2}{m} \boldsymbol{X}^T (\boldsymbol{X} \cdot \boldsymbol{\theta} - \boldsymbol{y}) \tag{2.11}$$

}

- Имея вектор-градиент, который указывает вверх, вы просто двигаетесь в противоположном направлении вниз. Это означает вычитание $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$ из $\boldsymbol{\theta}$. Именно здесь в игру вступает скорость обучения η : умножение вектора-градиента на η дает размер шага вниз в (2.11).

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Пакетный градиентный спуск***
  - Обратите внимание, что формулы (2.8) и (2.10) включают в себя вычисления с полным обучающим набором X на каждом шаге градиентного спуска. Вот почему алгоритм называется пакетным градиентным спуском (batch gradient descent): на каждом шаге он потребляет целый пакет обучающих данных.
  - В результате он оказывается крайне медленным на очень крупных обучающих наборах (но ниже мы рассмотрим гораздо более быстрые алгоритмы градиентного спуска).
  - Однако градиентный спуск хорошо масштабируется в отношении количества признаков; обучение линейной регрессионной модели при наличии сотен тысяч признаков проходит намного быстрее с применением градиентного спуска, чем с использованием нормального уравнения.

2 Обучение с учителем

2.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- ***Обучение на основе метода градиентного спуска. Пакетный градиентный спуск***
- Давайте рассмотрим быструю реализацию данного алгоритма:

```
eta = 0.1  # learning rate
n_iterations = 1000
m = 100 #Dataset size
# random initialization
theta = np.random.randn(2,1)  # random initialization
for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - eta * gradients
print("Найденный вектор параметров ЛР: \n", theta)
#Predicting
y_predict_gd=X_new_b.dot(theta)
print("Значения прогноза линейной регрессии: \n", y_predict)
rmse_lr = np.sqrt(mean_squared_error(y_target,y_predict))
r2_lr = r2_score(y_target,y_predict)
print("RMSE for LR:", rmse_lr)
print("R2_score for LR:", r2_lr)
```

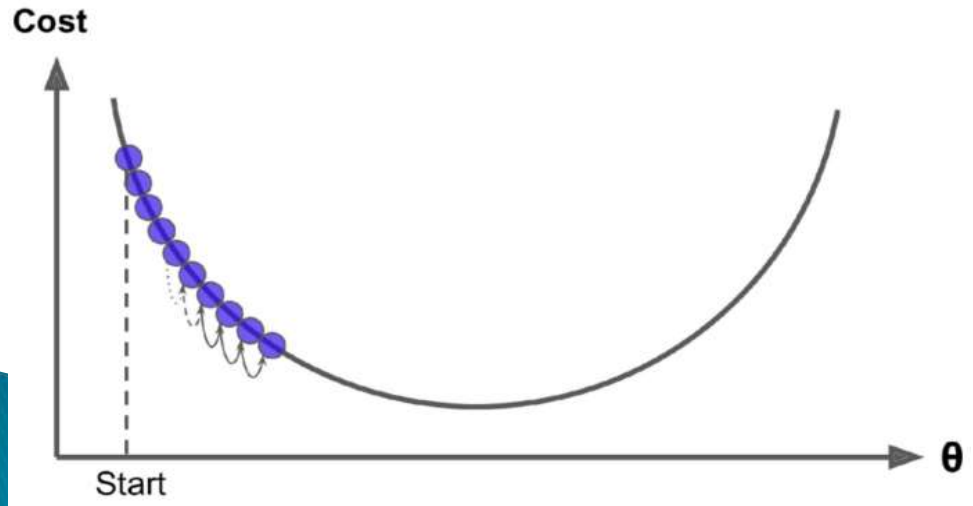
```
Найденный вектор параметров ЛР:
 [[3.75066965]
 [3.14047097]]
Значения прогноза линейной регрессии:
 [[ 3.75066965]
 [10.03161159]]
RMSE for LR: 0.1771145400549895 7
R2_score for LR: 0.996490838028116
```

- Результат совпадает с найденным аналитическим методом нормального уравнения и реализацией методами пакета Scikit-Learn. Градиентный спуск работает.

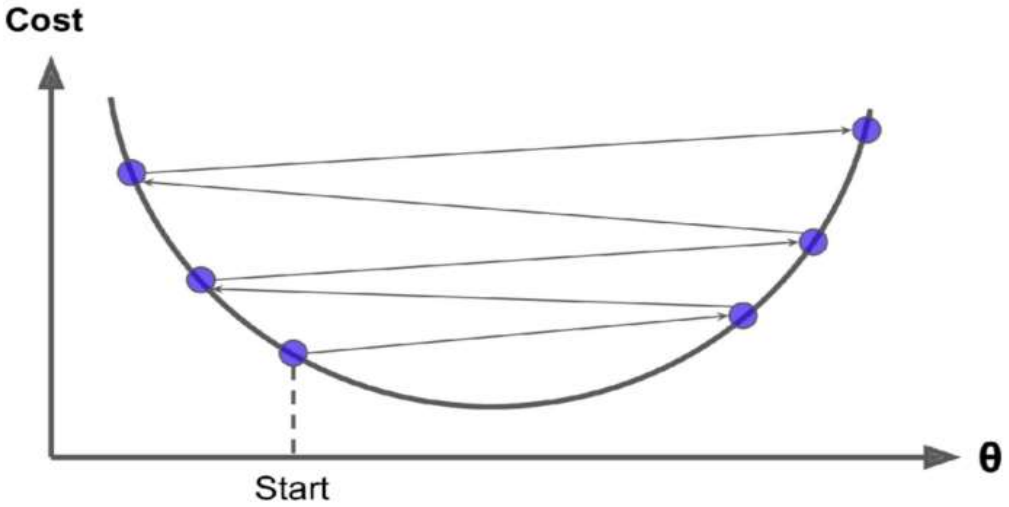
1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- ***Обучение на основе метода градиентного спуска. Скорость обучения***
- Важным параметром в градиентном спуске является размер шагов, определяемый гиперпараметром *скорости обучения* (*learning rate*) – параметром ηв формулах (2.9) и (2.11).
- Если скорость обучения слишком мала, тогда алгоритму для сведения придется пройти через множество итераций, что потребует длительного времени (левый рисунок 1).
- С другой стороны, если скорость обучения слишком высока, тогда можно перескочить долину и оказаться на другой стороне, возможно даже выше, чем находились ранее. Это способно сделать алгоритм расходящимся, что приведет к выдаче постоянно увеличивающихся значений и неудаче в поиске хорошего решения (правый рисунок 2).



1.Скорость обучения слишком мала

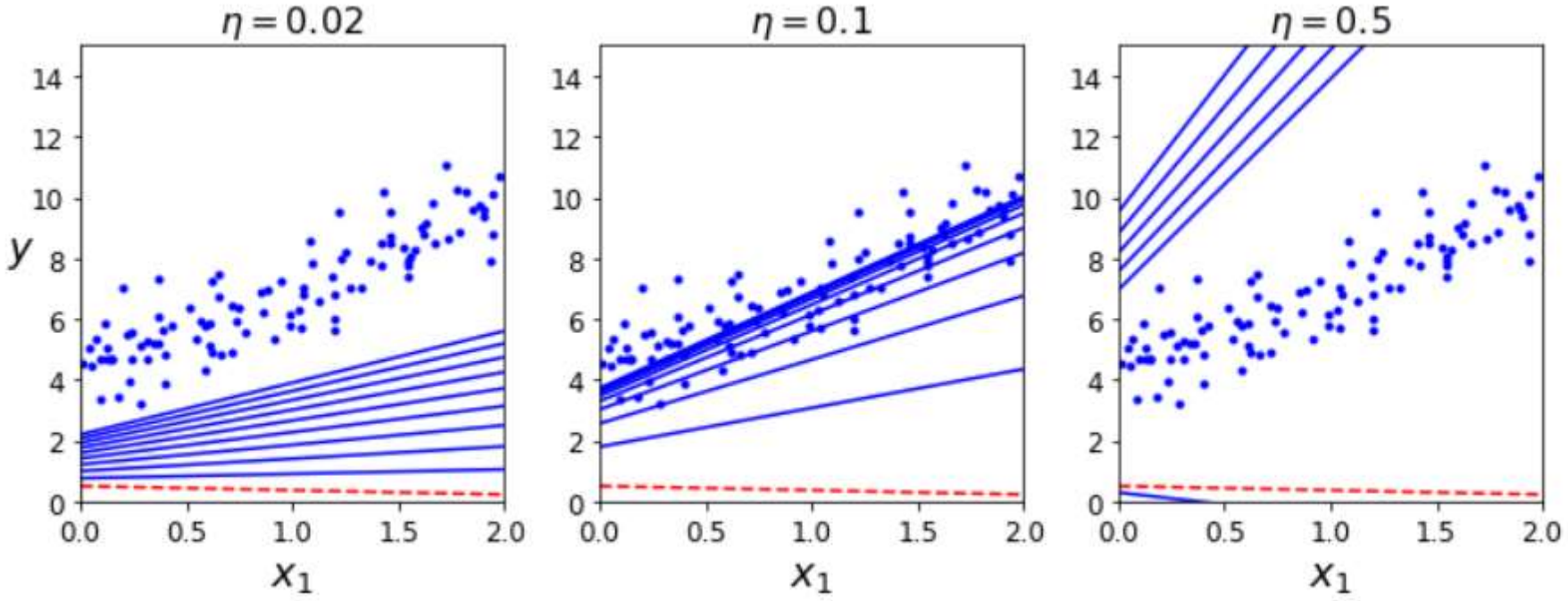


2.Скорость обучения слишком велика

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- ***Обучение на основе метода градиентного спуска. Скорость обучения***
- Давайте поэкспериментируем со скоростью обучения в обучении модели линейной регрессии для рассмотренного ранее примера. На рисунках ниже показаны первые 10 шагов градиентного спуска, использующих три разных скорости обучения (пунктирная линия представляет начальную точку).
- Слева скорость обучения слишком низкая: в конце концов, алгоритм достигнет решения, но это займет долгое время. Посредине скорость обучения выглядит довольно хорошей: алгоритм сходится к решению всего за несколько итераций (эта скорость использовалась нами при обучении выше). Справа скорость обучения чересчур высокая: алгоритм расходится, беспорядочно перескакивая с места на место и фактически с каждым шагом все больше удаляясь от решения.

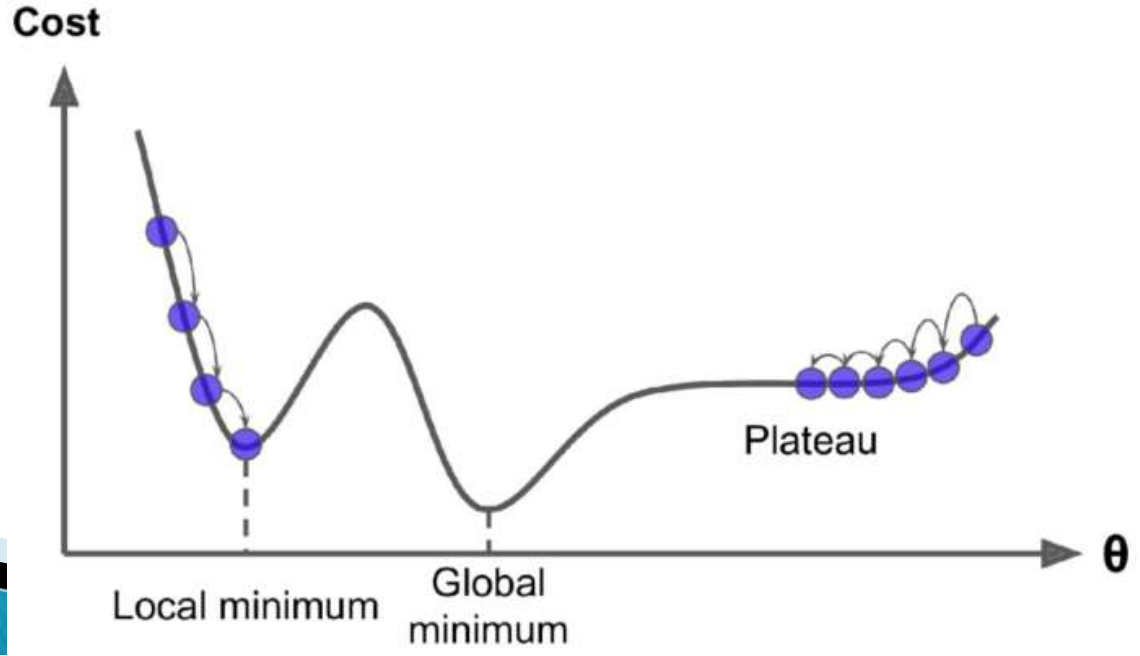


1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- ***Обучение на основе метода градиентного спуска. Функции издержек со сложным ландшафтом***

Не все функции издержек выглядят как точные правильные чаши. Могут существовать впадины, выступы, плато и самые разнообразные участки нерегулярной формы, которые крайне затрудняют схождение к минимуму. На рисунке ниже проиллюстрированы две главные проблемы с градиентным спуском: если случайная инициализация начинает алгоритм слева, то он сойдется в точке локального минимума, который не настолько хорош как глобальный минимум. Если алгоритм начнется справа, тогда он потратит очень долгое время на пересечение плато и в случае его слишком ранней остановки глобальный минимум никогда не будет достигнут.



1 Обучение с учителем

1.2 Линейная регрессия

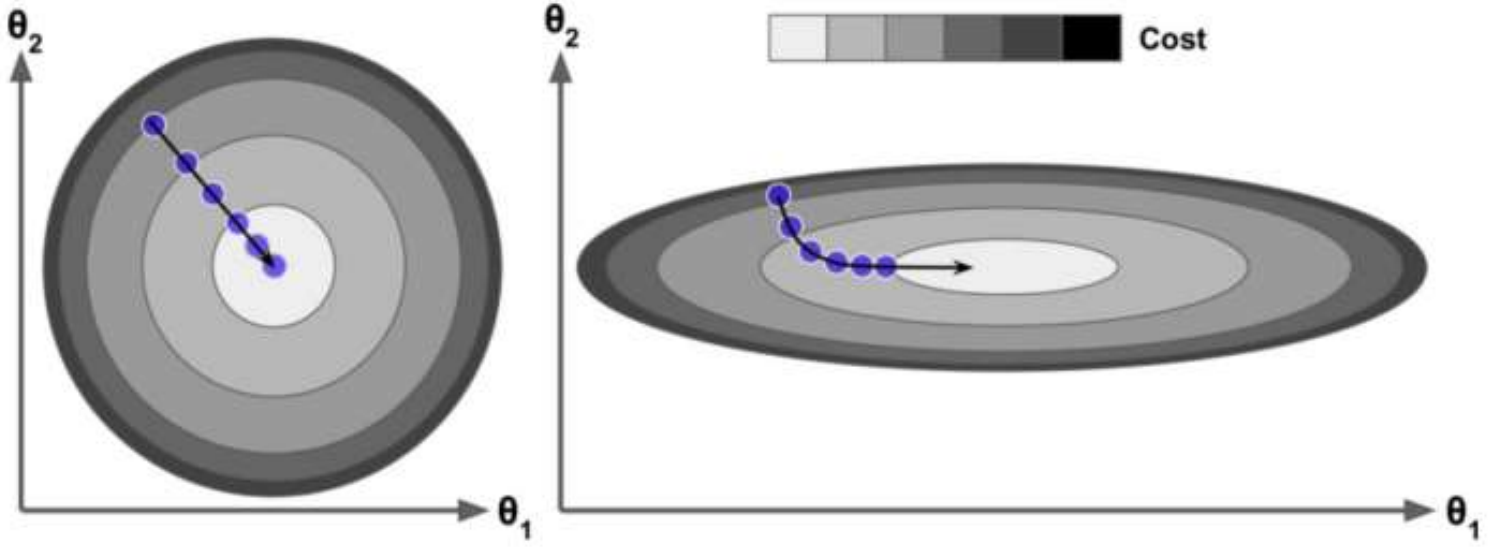
- ***Обучение модели линейной регрессии***
- ***Обучение на основе метода градиентного спуска. Функции издержек со сложным ландшафтом***

  К счастью, функция издержек MSE для линейной регрессионной модели является выпуклой функцией (convex function), т.е. если выбрать любые две точки на кривой, то соединяющий их отрезок прямой никогда не пересекает кривую. Отсюда следует, что локальные минимумы отсутствуют, а есть только один глобальный минимум. Она также представляет собой непрерывную функцию (continuous function) с наклоном, который никогда не изменяется неожиданным образом. Упомянутые два факта имеют большое значение: градиентный спуск гарантированно подберется произвольно близко к глобальному минимуму (если вы подождете достаточно долго и скорость обучения не слишком высока).

1 Обучение с учителем

1.2 Линейная регрессия

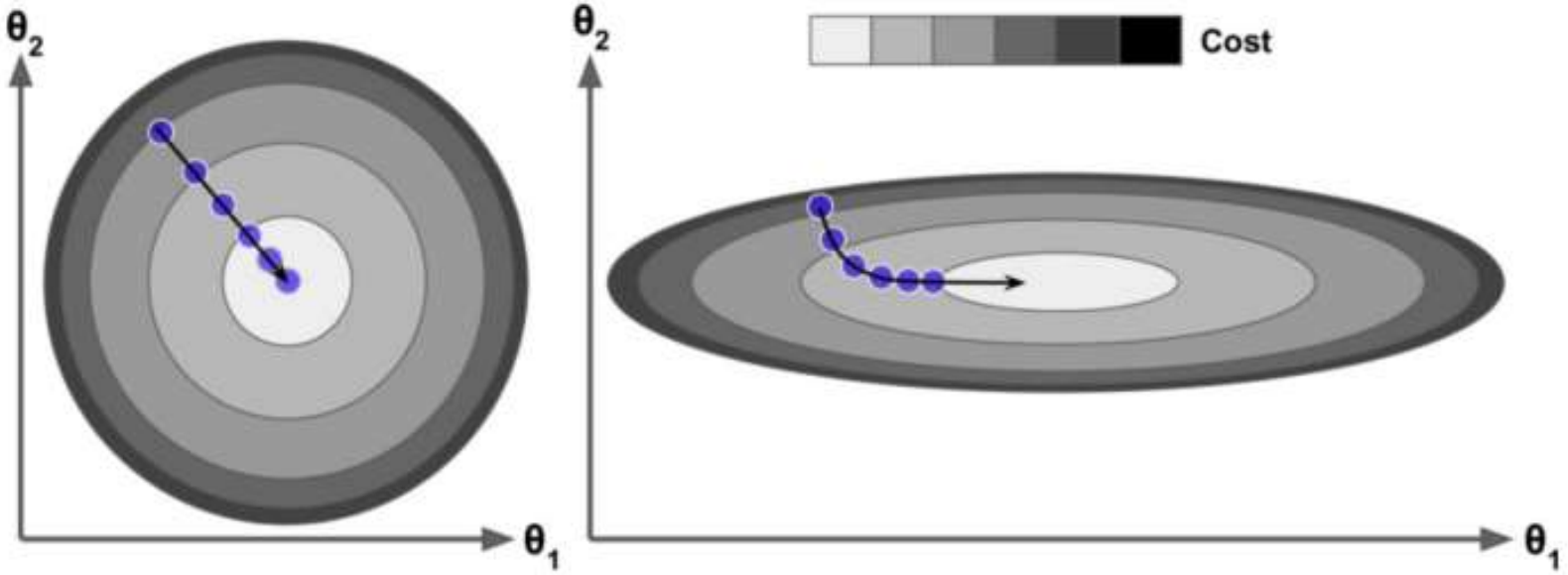
- ***Обучение модели линейной регрессии***
- ***Обучение на основе метода градиентного спуска. Нормализация данных***
- На самом деле функция издержек имеет форму чаши, но может быть продолговатой чашей, если масштабы признаков сильно отличаются. На рисунке показан градиентный спуск на обучающем наборе, где признаки 1 и 2 имеют тот же самый масштаб (слева), и на обучающем наборе, где признак 1 содержит гораздо меньшие значения, чем признак 2 (справа).
- Как видите, слева алгоритм градиентного спуска устремляется прямо к минимуму, из-за чего достигает его быстро, а справа он сначала двигается в направлении, которое практически перпендикулярно направлению к глобальному минимуму, и заканчивает длинным маршем по почти плоской долине. Минимум в итоге достигается, но за долгое время.



1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- ***Обучение на основе метода градиентного спуска. Нормализация данных***
- Таким образом, когда используется градиентный спуск, вы должны обеспечить наличие у всех признаков похожего масштаба (например, с применением класса StandardScaler из Scikit-Learn), иначе он потребует гораздо большего времени на схождение.
- Графики также демонстрируют тот факт, что обучение модели означает поиск комбинации параметров модели, которая сводит к минимуму функцию издержек (на обучающем наборе). Это поиск в пространстве параметров модели: чем больше параметров имеет модель, тем больше будет измерений в пространстве параметров и тем труднее окажется поиск
- Искать иглу в 300-мерном стоге сена намного сложнее, чем в трехмерном. К счастью, поскольку функция издержек выпуклая в случае линейной регрессии, иголка находится просто на дне чаши.



1 Обучение с учителем

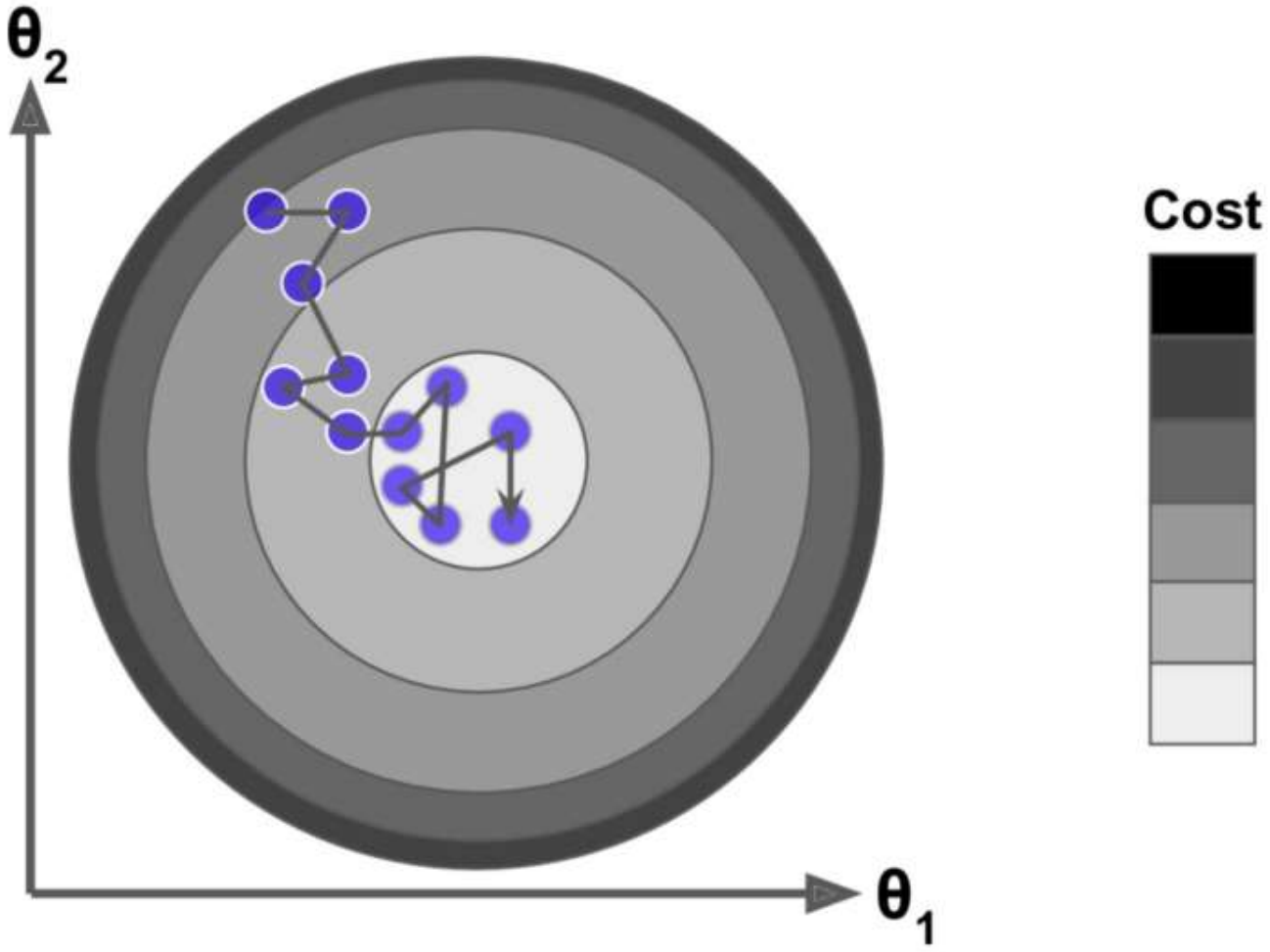
1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- ***Обучение на основе метода градиентного спуска. Стохастический градиентный спуск***
- Главная проблема с пакетным градиентным спуском – тот факт, что он использует полный обучающий набор для вычисления градиентов на каждом шаге, который делает его очень медленным в случае крупного обучающего набора.
- Как противоположная крайность, *стохастический градиентный спуск* (*Stochastic Gradient Descent - SGD*) на каждом шаге просто выбирает из обучающего набора случайный образец и вычисляет градиенты на основе только этого единственного образца. Очевидно, алгоритм становится гораздо быстрее, т.к. на каждой операции ему приходится манипулировать совсем малым объемом данных. Также появляется возможность проводить обучение на гигантских обучающих наборах, потому что на каждой итерации в памяти должен находиться только один образец.
- С другой стороны, из-за своей стохастической (т.е. случайной) природы этот алгоритм гораздо менее нормален, чем пакетный градиентный спуск:
  - ✓ вместо умеренного понижения вплоть до достижения минимума функция издержек будет скачками изменяться вверх и вниз, понижаясь только в среднем;
  - ✓ со временем алгоритм будет очень близок к минимуму, но как только он туда доберется, скачкообразные изменения продолжатся, никогда не успокаиваясь;
  - ✓ таким образом, после окончания алгоритма финальные значения параметров оказываются хорошими, но не оптимальными.

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
- ***Обучение на основе метода градиентного спуска. Стохастический градиентный спуск***
- При стохастическом градиентном спуске каждый шаг обучения выполняется намного быстрее, но также гораздо более стохастически, чем при использовании пакетного градиентного спуска.



1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Стохастический градиентный спуск***

- Следовательно, случайность хороша для избегания локальных оптимумов, но плоха потому, что означает, что алгоритм может никогда не сойтись к минимуму.
- Одним из решений такой дилеммы является постепенное сокращение скорости обучения. Шаги начинаются с больших (которые содействуют в достижении быстрого прогресса и избегании локальных минимумов), а затем становятся все меньше и меньше, позволяя алгоритму обосноваться на глобальном минимуме. Такой процесс называется *имитацией отжига* (*simulated annealing*), поскольку он имеет сходство с процессом закалки в металлургии, когда расплавленный металл медленно охлаждается.
- Функция, которая определяет скорость обучения на каждой итерации, называется *расписанием обучения* (*learning schedule*).
- Если скорость обучения снижается слишком быстро, то вы можете застрять в локальном минимуме или даже остаться на полпути к минимуму. Если скорость обучения снижается чересчур медленно, тогда вы можете долго прыгать возле минимума и в конечном итоге получить квазиоптимальное решение в случае прекращения обучения слишком рано.

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Стохастический градиентный спуск***
- Приведенный ниже код реализует стохастический градиентный спуск с применением простого графика обучения:

```
theta_path_sgd = []
m = len(X_b)
np.random.seed(42)
# Epoch number
n_epochs = 50
# learning schedule hyperparameters
t0, t1 = 5, 50

# Schedule of learning rate
def learning_schedule(t):
    return t0 / (t + t1)

# random initialization
theta = np.random.randn(2,1)
```

```
for epoch in range(n_epochs):
    for i in range(m):
        if epoch == 0 and i < 20:
            y_predict = X_new_b.dot(theta)
            style = "b-" if i > 0 else "r--"
            plt.plot(X_new, y_predict, style)
        random_index = np.random.randint(m)
        xi = X_b[random_index:random_index+1]
        yi = y[random_index:random_index+1]
        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(epoch * m + i)
        theta = theta - eta * gradients
        theta_path_sgd.append(theta)
```

```
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.axis([0, 2, 0, 15])
plt.show()
```

- По соглашению мы повторяем раундами по *m* итераций. Каждый раунд называется эпохой (epoch). Наряду с тем, что алгоритм пакетного градиентного спуска должен повторяться 1000 раз на всем обучающем наборе, показанный выше код проходит через обучающий набор только 50 раз.

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Стохастический градиентный спуск***
- При этом алгоритм стохастического градиента добивается относительно приемлемого решения.

```
print("Найденный вектор параметров ЛР: \n", theta)
#Predicting
print("Значения прогноза линейной регрессии: \n", y_predict)
rmse_lr = np.sqrt(mean_squared_error(y_target,y_predict))
r2_lr = r2_score(y_target,y_predict)
print("RMSE for LR:", rmse_lr)
print("R2_score for LR:", r2_lr)
```

```
Найденный вектор параметров ЛР:
 [[3.71930779]
 [3.1742288 ]]
Значения прогноза линейной регрессии:
 [[3.24832163]
 [9.46162788]]
RMSE for LR: 0.6537831897407215
R2_score for LR: 0.9525075045347164
```

- Обратите внимание, что поскольку образцы выбираются случайно, некоторые из них могут быть выбраны несколько раз за эпоху, тогда как другие – вообще не выбираться. Если вы хотите обеспечить, чтобы алгоритм в рамках каждой эпохи проходил по каждому образцу, то другой подход предусматривает тасование обучающего набора, проход по нему образец за образцом, снова тасование и т.д. Однако при таком подходе схождение обычно происходит медленнее.

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Стохастический градиентный спуск***
- Чтобы выполнить линейную регрессию, использующую SGD, с помощью Scikit-Learn, вы можете применять класс SGDRegressor, который по умолчанию настроен на оптимизацию функции издержек в виде среднеквадратичной ошибки.
- Следующий код прогоняет 50 эпох, начиная со скорости обучения 0.1 (eta0=0.1), но не применяет какой-либо регуляризации (penalty=None):

```
from sklearn.linear_model import SGDRegressor
#Fittin LR model with SGD method
sgd_reg = SGDRegressor(max_iter=1000, tol=1e-3, \
                       penalty=None, eta0=0.1, random_state=42)
lin_regSGD = sgd_reg.fit(X, y.ravel())
#Predicting
y_predict = lin_regSGD.predict(X_new)
y_target= 4 + 3 * X_new
print("Найденный вектор параметров ЛР: \n", theta)
print("Значения прогноза линейной регрессии: \n", y_predict)
```

```
Найденный вектор параметров ЛР:
 [[3.71930779]
 [3.1742288 ]]
Значения прогноза линейной регрессии:
 [ 3.71849147 10.04813949]
```

- И снова вы находите решение, очень близкое к тому, что возвращалось в ручную реализованным SGD.

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Мини-пакетный градиентный спуск***

- *Мини-пакетный градиентный спуск (mini-batch gradient descent)* понять довольно просто, т.к. мы знаем пакетный и стохастический градиентные спуски.
- На каждом шаге вместо вычисления градиентов на основе полного обучающего набора (как в пакетном градиентном спуске) или только одного образца (как в стохастическом градиентном спуске) мини-пакетный градиентный спуск вычисляет градиенты на небольших случайных наборах образцов, которые называются *мини-пакетами (mini-batch)*.
- Главное превосходство мини-пакетного градиентного спуска над стохастическим градиентным спуском в том, что вы можете получить подъем производительности от аппаратной оптимизации матричных операций, особенно когда используются графические процессоры.
- Продвижение этого алгоритма в пространстве параметров не настолько нерегулярно, как в случае SGD, в особенности при довольно крупных минипакетах. В результате мини-пакетный градиентный спуск закончит блуждания чуть ближе к минимуму, чем SGD. Но с другой стороны ему может быть труднее уйти от локальных минимумов (в случае задач, которые страдают от локальных минимумов).

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Мини-пакетный градиентный спуск***

Разделим выборку из m элементов на пакеты по 1000 элементов:

$$X = \{X^{\{1\}} = x^{(1)}, x^{(2)}, \ldots, x^{(1000)} | X^{\{2\}} = x^{(1001)}, x^{(1002)}, \ldots, x^{(2000)} | \ldots | X^{\{5000\}} = \ldots, x^{(m)}\}$$

Размер каждого пакета $X^{\{i\}}(n_x, 1000)$

Также следует разделить и выходные значения:

$$Y = \{Y^{\{1\}} = y^{(1)}, y^{(2)}, \ldots, y^{(1000)} | Y^{\{2\}} = y^{(1001)}, y^{(1002)}, \ldots, y^{(2000)} | \ldots | Y^{\{5000\}} = \ldots, y^{(m)}\}$$

Размер каждого пакета $Y^{\{i\}}(1, 1000)$

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Мини-пакетный градиентный спуск***

for $t = 1, \ldots, 5000$

Вычислить прямое распространение для $X^{\{t\}}$

$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$

$A^{[1]} = g^{[1]}(Z^{[1]})$

...

$A^{[L]} = g^{[L]}(Z^{[L]})$

Вычислить функцию стоимости $J^{\{t\}}$

Вычислить производные для $J^{\{t\}}$

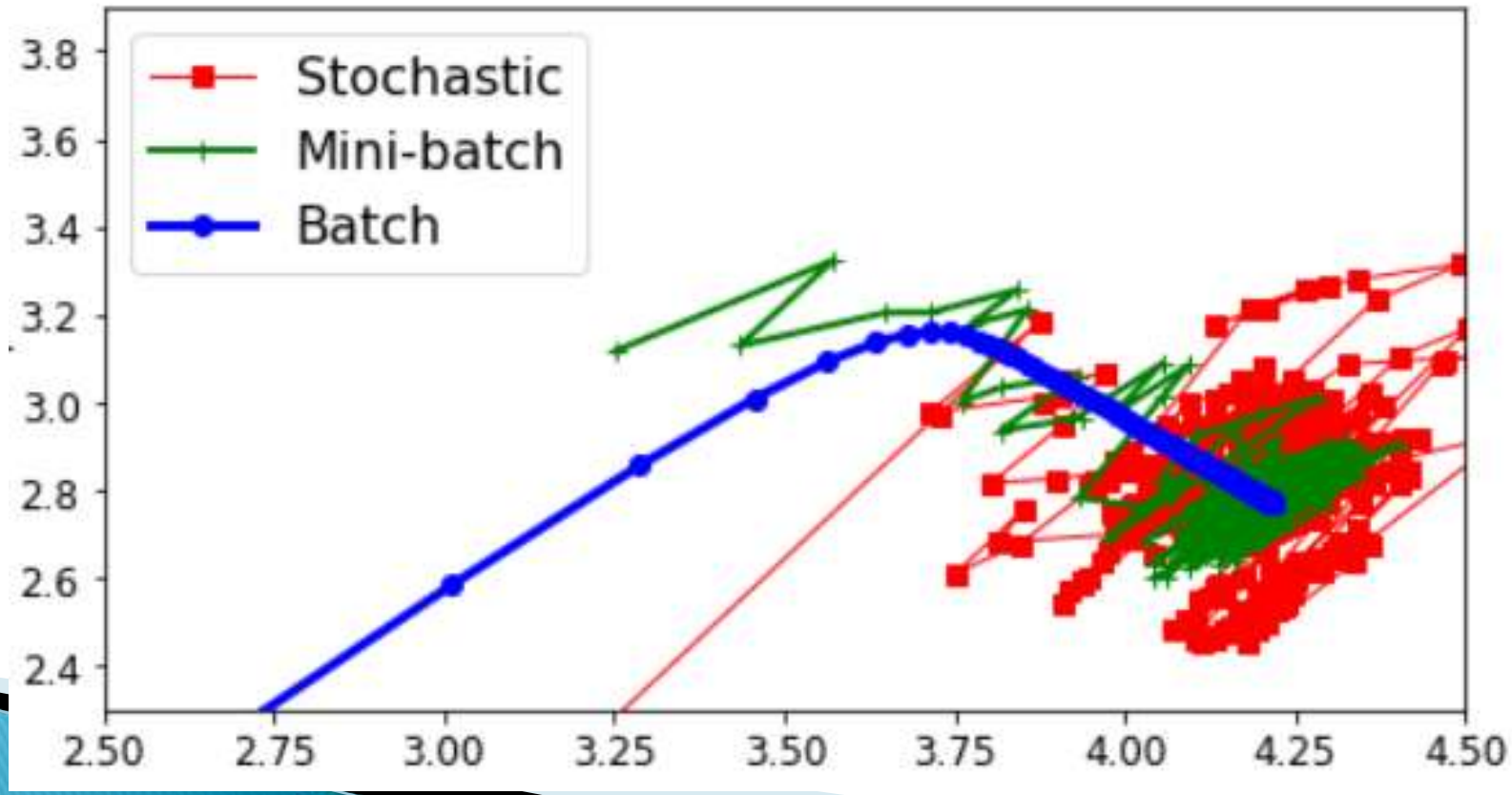
Обновить параметры $W^{[l]} = W^{[l]} - \alpha dW^{[l]}$, $b^{[l]} = b^{[l]} - \alpha db^{[l]}$

Проход полной выборки (5000 пакетов по 1000 элементов) называется “эпохой”

1 Обучение с учителем

1.2 Линейная регрессия

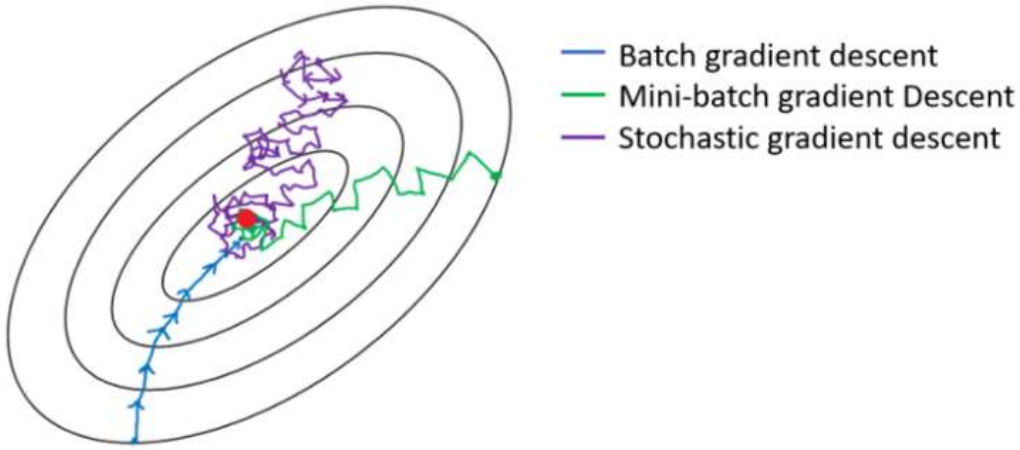
- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Мини-пакетный градиентный спуск***
- На рисунке ниже показаны пути, проходимые тремя алгоритмами градиентного спуска в пространстве параметров во время обучения. В итоге все они оказываются близко к минимуму, но путь пакетного градиентного спуска фактически останавливается на минимуме, тогда как стохастический и мини-пакетный градиентные спуски продолжают двигаться около минимума.



1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Мини-пакетный градиентный спуск***



Типичные размеры пакетов : 64 (2^6), 128 (2^7), 256 (2^8), 512 (2^9), ...;

Удостовериться, чтобы размер пакета укладывался в размер памяти CPU/GPU.

Стохастический градиентный спуск (размер пакета $= 1$) – не дает преимущества для векторизации
Пакетный градиентный спуск (размер пакета $= m$) – вычисление каждой итерации слишком большое
Мини-пакетный градиентный спуск (размер пакета $\ll m$) – получает преимущество от векторизации, а также каждая итерация вычисляется быстро

1 Обучение с учителем

1.2 Линейная регрессия

- ***Обучение модели линейной регрессии***
  - ***Обучение на основе метода градиентного спуска. Мини-пакетный градиентный спуск***
  - И снова вы находите решение, очень близкое к предыдущим результатам.

```
theta_path_mgd = []
# Epoch number
n_iterations = 50
# Minibatch size
minibatch_size = 20

# random initialization
np.random.seed(42)
theta = np.random.randn(2,1)
# learning schedule hyperparameters
t0, t1 = 200, 1000

def learning_schedule(t):
    return t0 / (t + t1)
```

```
t = 0
for epoch in range(n_iterations):
    shuffled_indices = np.random.permutation(m)
    X_b_shuffled = X_b[shuffled_indices]
    y_shuffled = y[shuffled_indices]
    for i in range(0, m, minibatch_size):
        t += 1
        xi = X_b_shuffled[i:i+minibatch_size]
        yi = y_shuffled[i:i+minibatch_size]
        gradients = 2/minibatch_size * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(t)
        theta = theta - eta * gradients
        theta_path_mgd.append(theta)
```

```
#Predicting
y_predict = X_new_b.dot(theta)
y_target= 4 + 3 * X_new
print("Найденный вектор параметров ЛР: \n", theta)
print("Значения прогноза линейной регрессии: \n", y_predict)
```

```
Найденный вектор параметров ЛР:
 [[3.67853559]
 [3.06988183]]
Значения прогноза линейной регрессии:
 [[3.67853559]
 [9.81829926]]
```

1 Обучение с учителем

1.3 Полиномиальная регрессия. Постановка задачи

- Что, если ваши данные в действительности сложнее обычной прямой линии? Удивительно, но вы на самом деле можете применять линейную модель для подгонки к нелинейным данным. Простой способ предполагает добавление степеней каждого признака в виде новых признаков и последующее обучение линейной модели на таком расширенном наборе признаков. Этот прием называется *полиномиальной регрессией (polynomial regression)*.

- *Гипотеза*

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

- *Признаки*

$$x_1 = x$$
$$x_2 = x^2$$
$$x_3 = x^3$$

- *Функция издержек (потерь)*

$$J(\boldsymbol{\theta}) = \mathrm{MSE}(\boldsymbol{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{\theta}^T \cdot x^{(i)} - y^{(i)})^2$$

- Необходимо решить задачу минимизации функции потерь

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$