## Парная линейная регрессия

### Цель работы

Познакомиться с основными понятиями задачи регрессии и метода градиентного спуска.

### Задания для выполнения

1. Загрузить данные о котировках нескольких инструментов с сайта РТС (следите за тем, чтобы периоды выгрузки совпадали, иначе весь наш анализ не будет иметь смысл).
   1. Можно взять не очень актуальные, но готовые данные [отсюда](https://drive.google.com/drive/folders/12RGy5CdYI9dXUkrLjzVXPYWmawbB1dU1?usp=sharing).
2. Построить модель парной линейной регрессии с использованием метода градиентного спуска.
3. Оценить на графике качество построенной модели.
4. Построить кривые обучения.

### Методические указания

Для полноценной работы с регрессионным анализом данных на потребуются следующие библиотеки языка Python:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

Для начала считаем данные о котировке двух инструментов из csv-файла, полученного при импорте с сайта биржи в объект Pandas DataFrame:

mvid = pd.read\_csv('data/MVID\_101001\_171001.txt')

sber = pd.read\_csv('data/SBER\_101001\_171001.txt')

Посмотрим формат файла с данными - состав и названия полей:

sber.head()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| <TICKER> | <PER> | <DATE> | <TIME> | <CLOSE> | <VOL> |  |
| 0 | SBER | D | 20101001 | 0 | 89.09 | 332429890 |
| 1 | SBER | D | 20101004 | 0 | 89.88 | 194032721 |
| 2 | SBER | D | 20101005 | 0 | 91.97 | 198775753 |
| 3 | SBER | D | 20101006 | 0 | 91.20 | 210406027 |
| 4 | SBER | D | 20101007 | 0 | 90.40 | 145137617 |

Если проанализировать два этих объекта внимательнее мы увидим, что они имеют разную длину - разное количество наблюдений. Как это может быть, если выгрузка производилась за одинаковый период времени? Проблема в том, что инструмент в определенные периоды может не торговаться. Поэтому, данные в каждом файле могут в части дней отсутствовать. Таким образом, метки времени в двух файлах не обязательно должны идти одинаково последовательно. Как раз для выявления таких проблем с данными и необходимо проводить первоначальный визуальный анализ данных.

Для разрешения нашей проблемы воспользуемся функцией merge из библиотеки pandas, которая и предназначена для соединения (в реляционном смысле) нескольких датасетов по ключевому полю:

res = pd.merge(sber, mvid, on='<DATE>')

res.head()

Затем выделим из получившегося объединенного датасета результативный и факторный признак. В нашем случае это будут цены закрытия двух этих инструментов:

x = res['<CLOSE>\_x']

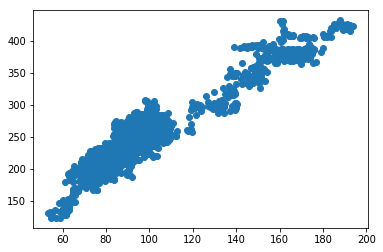
y = res['<CLOSE>\_y']

Для визуального представления совместного распределения значений используемых переменных построим график зависимости одной переменной от другой:

plt.figure()

plt.scatter(x, y)

plt.show()



Мы видим, что в общем и целом, линейная зависимость прослеживается в данных. Значит, мы можем продолжать и использовать парную линейную регрессию.

Начнем строить класс, который будет реализовывать метод парной линейной регрессии:

class hypothesis(object):

"""Модель парной линейной регрессии"""

def \_\_init\_\_(self):

self.b0 = 0

self.b1 = 0

Здесь мы определили конструктор класса, который запоминает в полях экземпляра параметры регрессии. В дальнейшем мы сможем расширять функциональность этого класса сколь угодно много.

реализуем метод, который принимает значение входной переменной и возвращает теоретическое значение выходной - это прямое действие нашей регрессии - метод предсказания результата по факторам (в случае парной регрессии - по одному фактору):

def predict(self, x):

return self.b0 + self.b1 \* x

Теперь зададим функцию ошибки:

def error(self, X, Y):

return sum((self.predict(X) - Y)\*\*2) / (2 \* len(X))

В данном случае мы используем простую функцию ошибки - среднеквадратическое отклонение (mean squared error, MSE). Можно использовать и другие функции ошибки. Именно вид функции ошибки будет определять то, какой вид регрессии мы реализуем. Существует много разных вариаций простого алгоритма регрессии. О большинстве распространенных методах регрессии можно почитать в официальной документации sklearn.

Теперь реализуем метод градиентного спуска. Он должен принимать массив X и массив Y и обновлять параметры регрессии в соответствии в формулами градиентного спуска:

def BGD(self, X, Y):

alpha = 0.00005

dJ0 = sum(self.predict(X) - Y) /len(X)

dJ1 = sum((self.predict(X) - Y) \* X) /len(X)

self.b0 -= alpha \* dJ0

self.b1 -= alpha \* dJ1

О выборе конкретного значения alpha - скорости обучения мы еще поговорим позднее.

Давайте создадим объект регрессии и проверим начальное значение ошибки:

hyp = hypothesis()

print(hyp.predict(0))

print(hyp.predict(100))

J = hyp.error(x, y)

print("initial error:", J)

0  
0  
initial error: 36271.58344889084

Как мы видим, для начала оба параметра регрессии равны нулю. Конечно, такая модель не дает надежных предсказаний, но в этом и состоит метод градиентного спуска: начиная с любого решения мы постепенно его улучшаем и приходим к оптимальному решению.

Теперь изобразим нашу регрессию на графике:

X0 = np.linspace(60, 180, 100)

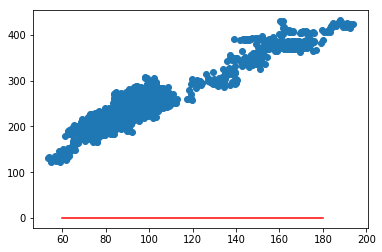
Y0 = hyp.predict(X0)

plt.figure()

plt.scatter(x, y)

plt.plot(X0, Y0, 'r')

plt.show()



Теперь все готово к запуску градиентного спуска.

hyp.BGD(x, y)

J = hyp.error(x, y)

print("error after gradient descent:", J)

error after gradient descent: 6734.135540194945

X0 = np.linspace(60, 180, 100)

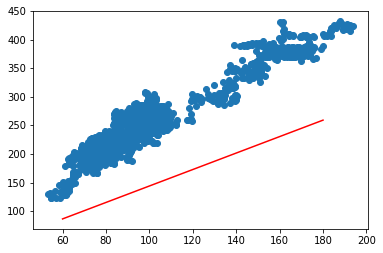
Y0 = hyp.predict(X0)

plt.figure()

plt.scatter(x, y)

plt.plot(X0, Y0, 'r')

plt.show()



Как мы видим, численное значение ошибки значительно уменьшилось. Да и линия на графике существенно приблизилось к точкам. Конечно, наша модель еще далека от совершенства. Мы прошли всего лишь одну итерацию градиентного спуска. Модифицируем метод так, чтобы он запускался в цикле пока ошибка не перестанет меняться существенно:

def BGD(self, X, Y, alpha=0.000005, accuracy=0.01):

steps, errors = [], []

step = 0

old\_err = hyp.error(X, Y)

new\_err = hyp.error(X, Y) - 1

dJ = 1

while dJ > accuracy:

dJ0 = sum(self.predict(X) - Y) /len(X)

dJ1 = sum((self.predict(X) - Y) \* X) /len(X)

self.b0 -= alpha \* dJ0

self.b1 -= alpha \* dJ1

old\_err = new\_err

new\_err = hyp.error(X, Y)

dJ = old\_err - new\_err

step += 1

steps.append(step)

errors.append(new\_err)

return steps, errors

Заодно мы расширили функциональность этого метода так, чтобы он возвращал массив ошибок на каждой итерации спуска и номер итерации. Это будет нам полезно для построения кривых обучения.

Запустим наш градиентный спуск:

hyp = hypothesis()

steps, errors = hyp.BGD(x, y)

J = hyp.error(x, y)

print("error after gradient descent:", J)

error after gradient descent: 298.76881676471504

Как мы видим, теперь ошибка снизилась гораздо больше. Однако, она все еще не достигла нуля. Заметим, что нулевая ошибка не всегда возможна в принципе из-за того, что точки данных не всегда будут располагаться на одной линии. Нужно стремиться не к нулевой, а к минимально возможной ошибке.

Посмотрим, как теперь наша регрессия выглядит на графике:

X0 = np.linspace(60, 180, 100)

Y0 = hyp.predict(X0)

plt.figure()

plt.scatter(x, y)

plt.plot(X0, Y0, 'r')

plt.show()

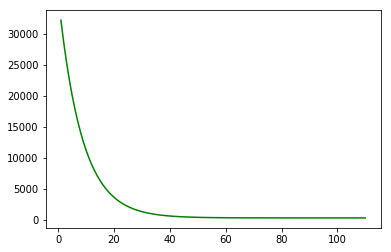
### 

Уже значительно лучше. Линия регрессии довольно похожа на оптимальную. Так ли это на самом деле, глядя на график, сказать сложно, для этого нужно проанализировать, как ошибка регрессии менялась со временем:

plt.figure()

plt.plot(steps, errors, 'g')

plt.show()



На графике наглядно видно, что в начале обучения ошибка падала быстро, но в ходе градиентного спуска она вышла на плато. Учитывая, что мы используем гладкую функцию ошибки второго порядка, это свидетельствует о том, что мы достигли локального оптимума и дальнейшее повторение алгоритма не принесет улучшения модели.

### Контрольные вопросы

1. Сформулируйте задачу регрессии.
2. Что такое метод наименьших квадратов?
3. Какие функции ошибки используются в регрессионных моделях? Расскажите про три самые популярные.
4. Что показывает конкретное значение ошибки регрессии?
5. Зачем нужно строить кривые обучения?

### Дополнительные задания

1. Реализуйте механизм адаптивной скорости обучения.
2. Постройте модель регрессии на другой паре инструментов.
3. Модифицируйте класс регрессии так, чтобы он проводил обучение только на случайно выделенной части данных (обучающей выборке). Историю динамики ошибки отслеживать отдельно по обучающей и по тестовой выборке.
4. Научиться использовать библиотеку sklearn для построения парной линейной регрессии:
   1. Изучить официальную документацию библиотеки sklearn: <http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html>
   2. По примеру программы регрессии, рассмотренной на семинаре, построить модель регрессии с использованием библиотечных средств
   3. Сравнить результаты регрессии с использованием библиотечной функции и написанной самостоятельно.