Сжатие изображений с помощью полиномиальной регрессии

Ранее мы рассматривали задачу регрессии как способ описать данные с помощью простой математической формулы. Мы ограничивались простейшим вариантом – линейной регрессией, когда зависимость выходной целевой величины от входных величин описывается уравнением прямой (или гиперплоскости в случае нескольких входных величин).

Сегодня мы рассмотрим ещё один интересный пример задачи регрессии – сжатие изображений с потерями с помощью полиномиальной регрессии.

В этой практике мы познакомимся с одной очень интересной и популярной библиотекой машинного обучения для Python – **`scikit-learn`**. Библиотека позволяет решать многие задачи Data Mining: классификацию, кластеризацию, регрессию, снижение размерности, подготовку данных и т.д. В библиотеке есть реализации известных нам алгоритмов: k-средних, k-ближайших соседей, деревья решений, искусственные нейронные сети и множество других популярных алгоритмов. При этом библиотека очень проста в использовании.

Рассмотрим для простоты монохромное изображение. Для кодирования оттенка отдельной точки такого изображения требуется 1 байт или 8 бит информации. Это позволяет сохранять 256 оттенков серого цвета для каждого пиксела изображения. Можно было бы существенно сократить объём изображения, если бы на кодирование каждого пиксела требовалось, например, 4 бита вместо 8. Этого можно добиться с использованием полиномиальной регрессии. В основе лежит очень простая идея. Можно рассмотреть отдельную строку изображения как массив целых чисел, а затем описать этот массив с помощью формулы – полинома n-й степени (см. рисунок 1).

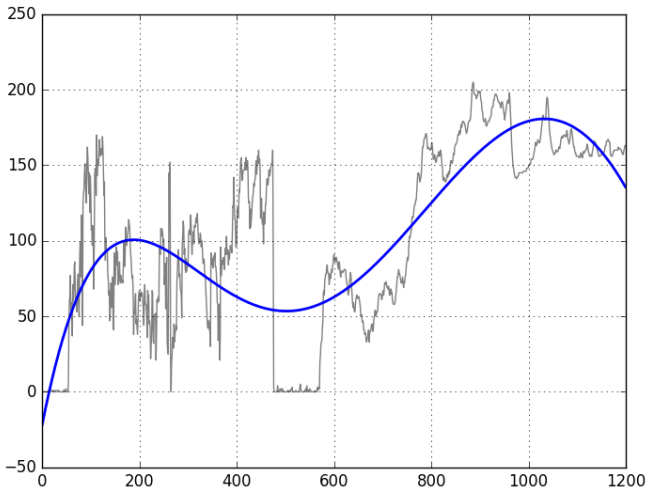

Рисунок 1 – Описание строки изображения с помощью полинома 5 степени

Таким образом, строку изображения из 1200 точек можно описать с помощью n+1 чисел-коэффициентов. На рисунке 1 используется полином 5 степени, то есть нужно 6 чисел, чтобы описать строку изображения. Но из рисунка также видно, что между реальными значениями оттенков строки изображения и кривой существует разница, которую уже можно описать с помощью меньшего числа бит. Например, если для кодирования разностей мы будем использовать 4 бита, то можно закодировать разности в 8 оттенков (1 бит кодирует знак – плюс или минус, 3 бита кодируют разность: $2^3 = 8$). Конечно, даже из рисунка 1 видно, что иногда эти разности значительно превосходят 8 оттенков, но их приходится укладывать в эти 8 оттенков. Отсюда и сжатие с потерями.

Порядок выполнения работы

Практика состоит из двух частей. Первая часть – общая для всех вариантов. Вторая часть – дополнительное задание по вариантам.

Открытие изображения

С помощью класса **Image** из библиотеки **Pillow** (**PIL**) откройте изображение. Сохраните оттенки цветов изображения в 3-мерный массив **NumPy**:

```
im = Image.open('путь_до_изображения')
data = np.array(im.getdata()).reshape([im.height, im.width, 3])
```

Не забудьте подключить необходимые библиотеки:

```
from PIL import Image
import numpy as np
```

Создание матрицы входных признаков

Создайте матрицу входных признаков с помощью приведённого ниже кода:

```
x = np.arange(0, im.width)
X = np.array([x, x**2.0, x**3.0, x**4.0, x**5.0]).transpose()
```

Здесь используется полином 5 степени. По аналогии вы можете вводить полином произвольной степени и даже использовать математические функции, например, тригонометрические синус и косинус, логарифм и т.п.

Построение и визуализация полинома

Для построения полинома используйте библиотеку **scikit-learn**. Для визуализации – **matplotlib**. Подключите библиотеки в скрипт с помощью данного кода:

```
from sklearn import linear_model
import matplotlib.pyplot as plt
```

Для того, чтобы решить задачу регрессии, нам потребуется массив с оттенками отдельной строки изображения. Оттенки у нас хранятся в массиве **data**. Это трёхмерный массив. При обращении к элементу массива первый индекс описывает строку изображения, второй – столбец изображения (пиксел в строке), третий – цветовой канал (**R**ed, **G**reen или **B**lue). Поэтому для получения оттенков синего цвета первой строки изображения может быть использован следующий код:

```
y = data[0, :, 2]
```

Отобразите на одном графике все 3 цветовых канала. Должен получиться примерно такой график, как на рисунке 2.

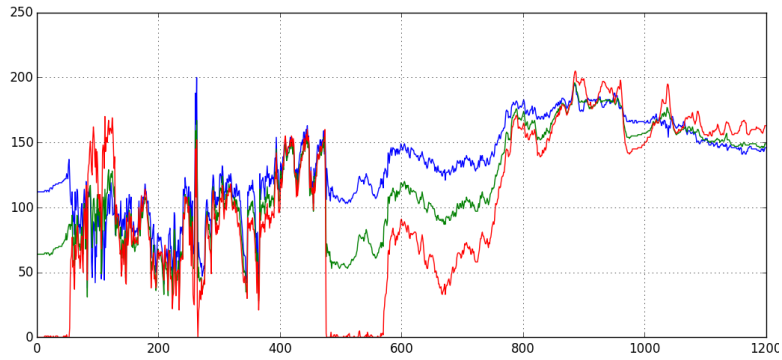

Рисунок 2 – Цветовые каналы первой строки изображения

У нас есть реальные значения оттенков строки изображения. Теперь нужно построить кривую, которая будет описывать эти оттенки. Для этого используйте приведённый ниже код:

```
lm = linear_model.LinearRegression()
lm.fit(X, y)
predicted = lm.predict(X)
```

Здесь мы используем линейную регрессию из библиотеки **scikit-learn** для построения кривой. Функция **fit** подбирает оптимальные коэффициенты для построения кривой, функция **predict** определяет значения целевой функции для заданных входных параметров **X**. Чтобы убедиться в правильности работы данного кода, отобразите на одном графике реальные (**y**) и предсказанные (**predicted**) значения. Должен получиться график, похожий на приведённый на рисунке 1.

Кодирование разностей

У нас имеется массив реальных значений оттенков (**y**) и массив оттенков, рассчитанных по формуле с помощью полинома (**predicted**). Они, конечно же, различаются. Необходимо вычислить эти разности и закодировать их.

Напишите код для вычисления разностей реальных и вычисленных по формуле оттенков.

Далее задайте, сколько бит потребуется для хранения разностей по каждой точке. Например, так:

```
bits_per_channel = 4
```

Теперь рассчитайте количество оттенков, которые можно закодировать с помощью данного числа бит. Например, если мы кодируем разности с помощью 4 бит, то один бит используется для кодирования знака (плюс или минус), а оставшиеся 3 бита для кодирования 8 оттенков от 0 до 7. Количество оттенков в разностях, которые можно закодировать данным числом бит, сохраните в переменной **threshold**. В данном примере это значение будет равно 7, но должно рассчитываться из переменной **bits_per_channel**.

Разности, которые выходят за пределы **threshold**, должны быть искусственно обрезаны до значений **threshold**. В этом и состоит сжатие с потерями. Чтобы отсечь значения, выходящие за порог, используйте функцию **clip** из библиотеки **NumPy**. Не забудьте, что отсекать нужно разности как больше порога, так и меньше с обратным знаком.

Теперь, после того, как разности были усечены до заданного числа бит, нужно выполнить обратную операцию и получить усечённые реальные значения оттенков. Для этого к значениям **predicted**, вычисленным по формуле, нужно прибавить усечённые разности (например, они хранятся в массиве **diff**):

```
y = predicted + diff
```

Подмена пикселов в исходном изображении

После того, как Вы вычислили значения оттенков на предыдущем этапе, необходимо записать их обратно в изображение. Чтобы работать с пикселями изображения, выполните код:

```
pix = im.load()
```

К каждому отдельному пикселю можно обратиться так (в квадратных скобках индекс строки и столбца изображения):

```
>>> pix[5, 10]
(0, 64, 112)
```

Результатом является **`tuple`** (кортеж), где каждое число отвечает за оттенок цветового канала (Red, Green и Blue соответственно).

Изменить значение отдельного канала пикселя можно так:

```
>>> pix[5, 10]
(0, 64, 112)
>>> l = list(pix[5, 10])
>>> l[0] = 10
>>> pix[5, 10] = tuple(l)
>>> pix[5, 10]
(10, 64, 112)
```

Теперь дополните Ваш скрипт так, чтобы кодирование выполнялось для каждой строки и каждого цветового канала изображения (используйте вложенные циклы) и сохраните получившееся изображение с помощью команды

```
im.save('ready.png')
```

Вот как выглядит результат сжатия с помощью полинома 5 степени с упаковкой каждого оттенка пикселя в 5 бит:

Рисунок 3 – Исходное изображение и упакованное по каждому пикселу до 5 бит

Конечно, качество изображения после сжатия получилось слишком низким, но использованный подход является слишком простым, и направлен скорее на знакомство с полиномиальной регрессией, нежели на реальное сжатие.

Задания по вариантам:

Вариант 1:

Привести графики как на рисунке 1 для всех цветовых каналов отдельной строки изображения. Должно быть 3 графика на одном объекте **`Figure`**, для каждого цветового канала должна быть рассчитана и изображена кривая регрессии.

Вариант 2:

Привести результаты кодирования изображения 3, 4, 5, 6 и 7 битами, а также исходное изображение.

Вариант 3:

Исследовать различные варианты полиномов: второй, третьей, четвёртой, пятой степеней. Для каждого из вариантов построить кривую регрессии и отобразить на одном графике вместе с реальными значениями (как на рисунке 1).